PEACH ROM NOTES

by Bruce Rossell and Howard Viccars

THE SOFTWARE HOUSE

Canberra A.C.T.

These notes have been prepared using the HiWriter word processing package (copyright Cybernetics Research) and printed on a Qume Sprint 5 printer.

Published by:

THE SOFTWARE HOUSE
P.O. Box 70
Weston A.C.T. 2611
Phone (062)885061 a.h.

TABLE OF CONTENTS

SECTION 1
INTRODUCTION

SECTION 2
NOTES ON HOW BASIC WORKS

SECTION 3
FILE HANDLING AND I/O

SECTION 4
KEYBOARD HANDLING

SECTION 5
SCREEN HANDLING AND GRAPHICS

SECTION 6
NOTES ON SPECIFIC COMMANDS

SECTION 7
IMPLEMENTING NEW COMMANDS

SECTION 8
APPARENT BUGS, ANOMALIES AND SOME "FIXES"

SECTION 9
USEFUL ROUTINES AND GENERAL NOTES

APPENDIX A
BASIC AND SYSTEM WORK AREA

APPENDIX B
I/O REGISTERS

APPENDIX C
BASIC JUMP TABLE

APPENDIX D
KEYBOARD

APPENDIX E
NOTES ON SSSD DISK BASIC

APPENDIX F
MEMORY DISPLAY AND SSSD DOSMOD PROGRAMS

1.1 BACKGROUND

1.1.1 These notes represent countless hours of experimentation with the Peach and investigation of a disassembly listing of the ROM. They are not intended as a complete explanation of the workings of the ROM. They are essentially a collection of the authors notes on specific aspects of the Peach.

1.1.2 The notes are intended to allow Peach owners to make better use of the routines within the ROM. It is hoped that they will form a basis for further experimentation and assist with increasing the volume of available software for the Peach.

1.2 LIABILITY

1.2.1 The authors and publisher assume no liability with respect to the use of this publication nor any damages arising from the use of any information contained herein. Specifically no liablity is accepted for errors in the routines listed in the notes.

1.2.2 The notes are sold on the understanding that they are not intended as a definitive manual on the workings of the ROM. They only represent the authors notes and due to the complexities of the ROM the interpretation of the code may in some cases be incorrect or misleading.

1.3 REQUIRED LEVEL OF "EXPERTISE"

1.3.1 The notes assume a basic level of understanding of the Peach and programming in general. Inexperienced users may be confused by some sections but should find the notes valuable overall. Many of the routines may be executed without an understanding of how they work.

1.3.1 Many of the standard features of the Peach have been assumed to be understood. The BASIC manual gives a reasonable explanation, despite the translation errors, and these notes have not included areas covered in the BASIC manual. Where relevant the reference to the specific section in the BASIC manual has been given.

1.3.2 All the machine code routines have been written and tested using the Hitachi Assembler/Debugger (ASM9). It is assumed that readers wishing to utilise these routines will have a basic understanding of machine code programming.

1.4 COPYRIGHT

1.4.1 These notes are subject to copyright, as indicated at the foot of each page, and may not be copied or reproduced in any form, including electronic forms, without the written permission of the publisher.

1.4.2 The notes represent countless hours of effort and the copyright should be respected. The notes have been specifically priced at a reasonable level to allow most Peach owners to purchase a copy.

1.5 ERRORS AND PROBLEMS

1.5.1 Whilst no liability is accepted for errors and problems arising from the use of the notes all reasonable attempts will be made to correct errors and misleading information. There is obviously a limit to the amount of time that may reasonably be spent on any one error and the authors reserve the right not to correct specific errors.

1.5.2 All errors, problems, and further information relating to the notes should be forwarded to the publisher at the address shown on the cover page. Any further information relating to the notes or the Peach in general will be appreciated and may be included in future editions. No payment will be made for this information unless specifically agreed in writing with the publisher.

1.6 AMENDMENTS AND FUTURE EDITIONS

1.6.1 Due to the nature of these notes it is likely that there will be errors and routines that do not work under certain conditions. Amendments to the notes will be published when the volume of corrections or additional information warrants it. The availability, and price, of amendments will be advertised in the Readers' Classifieds section of YOUR COMPUTER magazine.

1.6.2 Depending on the future of the Peach it is hoped that a second edition of these notes will be produced in the future. Purchasers of previous editions will be given the option of updating to the new edition at a discounted price provided the earlier edition is returned.

1.7 ORDERING THE MEMORY DISPLAY AND DOSMOD PROGRAMS

1.7.1 Appendix F details two programs that are available to purchasers of these notes. The programs have not been included with the notes due to the varied Disk Basic formats.

1.7.2 The programs are available on SSSD disk or cassette. The price includes the media cost together with postage and handling. An order form is included with the notes.

2.1 INTRODUCTION

2.1.1 The following notes outline the organisation of BASIC programs together with limited details on processing. The processing details for BASIC are complex and difficult to interpret from a disassembly listing of the ROM. A detailed understanding of how BASIC works is not necessary to utilise many of the routines within the ROM.

2.2 BASIC POINTERS

001D-001E	Start of BASIC
001F-0020	Start of Variables
0021-0022	Start of Arrays
0023-0024	Start of free space
0025-0026	Top of Stack - lower limit of string space
002B-002C	Top of String space
03D8-03D9	Top of available memory

2.2.1 The above pointers define the position of BASIC in relation to the available RAM. The values in these pointers are variable and depend on the screen mode, number of disk buffers, use of the CLEAR command, etc. Appendix A details the common values for locations $1D-1E, the start of BASIC.

2.2.2 The pointers to the start of variables, arrays and free space depend on the particular program in memory and its state of execution. BASIC allocates space for variables at the time they are first referenced and, therefore, these pointers may vary during the execution of a BASIC program.

2.2.3 The pointers to the start of string space and the top of string space depend on the use of the CLEAR command. The default values set up on initialisation will allocate a 300 byte string area starting at the top of available memory, allowing for the DOS if present. The stack is then located at the bottom of string space and grows downward towards the BASIC code. The CLEAR statement may reallocate both the length of string space and its location depending on the parameters specified.

2.2.4 The area between the start of free space (locations \$23-24) and the lower limit of string space (locations \$25-26) is available for machine code programs. A small amount of this space is used for the stack and therefore machine code programs should not be placed directly below string space - allow about 60 bytes. Location \$1B-1C is a copy of the stack pointer and should give an indication of the approximate size of the stack.

2.3 BASIC ORGANISATION

2.3.1 The standard organisation of a line of a BASIC program is as follows:-

PPPP	- two byte pointer to the start of the next line
NNNN	- line number of this line (two bytes)
BBBBBB...	- BASIC statement up to 255 bytes in length
00	- zero byte terminating the BASIC statement - followed by next line of BASIC or two zero bytes if last statement.

2.3.2 The above format is used for all lines within a BASIC program. Location \$1D-1E points to the start of the first line. The BASIC statement is terminated by a zero byte and may be up to 255 characters in length. BASIC commands are represented by tokens and are not stored in ASCII - refer Section 2.4.

2.3.3 The BASIC editor automatically inserts and deletes lines in the correct numerical order by moving part of the program up or down. The last statement of a BASIC program is terminated by a single byte (as detailed in the above paragraph) and then by two more zero bytes. These last two bytes are pointed to by the first two bytes of the last statement, i.e., when BASIC uses its pointers to look for the start of the next line and finds two zero bytes then it knows the end of the BASIC program has been reached.

2.4 BASIC TOKENS

2.4.1 Appendix C details the tokens corresponding to each BASIC statement and function. When a line of BASIC is input from the screen or an ASCII file the BASIC commands are translated into one or two byte tokens. Each word, except those enclosed within quotes, is checked against the command and function tables and convert to the corresponding token if a match is found.

2.4.2 The numerical value of the token is defined by its relative position within the command and function tables. All tokens have the top bit set which allows BASIC to differentiate between tokens and other characters. Graphic characters or any ASCII characters with a value greater than 127 are not valid when entering a line of BASIC and will cause a "Syntax Error" message.

2.4.3 The difference between a binary and ASCII save of a BASIC program relates to the tokens. In a binary save (the normal SAVE command without the ",A" paramater) the program is saved in the above format and includes tokens for the BASIC commands. In an ASCII save (using the ",A" parameter or the LIST command) the tokens are converted back to the full ASCII representation before being written to the output device. Therefore the ASCII save requires more space on the output device and will take longer to save, especially to cassette. The ASCII format is useful for moving programs to and from other systems.(+ for MERGE)

2.5 COMMAND AND FUNCTION TABLES

2.5.1 BASIC maintains tables for all commands and functions. These sets of tables have corresponding lists of addresses which define the start of the routine within the ROM which handles the particular command. The BASIC command table starts at location $A078 and the corresponding addresses start at location $A281. The BASIC function table starts at location $A1D9 and the corresponding addresses start at location $A000. Appendix C details these addresses for each command and in some cases includes a more relevant address where the command is actually handled.

2.5.2 Locations $0121-013D define the addresses for the above command and function tables and also for the Disk Basic command and function tables. In addition, it is possible to define a separate set of tables for additional commands - refer to Section 7 for further details.

2.6 BASIC'S ACCUMULATOR

0006		Accumulator variable type - 2=int 3=char 4=single 8=double
0056-005D		BASIC's accumulator (3-1-3 of BASIC manual)
005E-005F	??	Negative indicator

2.6.1 Location $06 indicates the type of the variable in the accumulator. Section 3-1-3 of the BASIC manual defines the format of the accumulator for the different variable types. The exact meaning of locations $5E-5F has not been determined, however, it appears to be a negative indicator.

2.6.2 Section 6.4 details some routines within the ROM that may be used to convert a numeric value in the accumulator from one variable type to another.

2.7 BASIC'S PROCESSING CYCLE

2.7.1 The details on BASIC's processing are complex and are not considered necessary to utilise many of the ROM routines. The following notes are only intended as a limited outline of the main processing cyle.

2.7.2 BASIC uses the $EF4B screen editor routine (similar to the $EF55 routine detailed in Sections 4.3 and 4.6.4) for input and only returns to the BASIC processing code when 'RETURN' or 'BREAK' have been pressed. The main processing code starts at location $A440. The instructions before this print the 'Ready' message.

2.7.3 The code around $A440 checks the command input after 'RETURN' to see if it is a BASIC statement (starts with a line number) or a direct statement. If it is a BASIC statement then it inserts it into the current program in numerical order. If it is a direct statement then it jumps to the command handler routine at $D3F7.

2.7.4 On completion of a command within a BASIC program the command handler returns to location $D3BA. This code checks location $03D5 to see if COMO, Light Pen or PF Key input has occurred and processes them if the ON COM/PEN/KEY command has been executed - the routine to handle ON COM/PEN/KEY starts at $EB1E. The $D3BA routine checks that the line of BASIC has been completed or is terminated by a ":" delimiter - if not it generates a "Syntax Error" message. Following this the next command is executed or control is returned to the BASIC command level ($A440).

2.7.5 In summary, the main loop for executing commands within a BASIC program is in the area $D3BA to $D45B. The BASIC command level is handled by the $EF4B input routine (called from $A443) until a 'RETURN' is entered. Once 'RETURN' has been entered the code around $A450 checks the type of command input and jumps to the command handler at $D3F7 if it is a direct command or inserts it into the current BASIC program if not (the code between $A467 and $A4CE handles the updating of a BASIC program).

2.8 EXECUTING BASIC COMMANDS FROM MACHINE CODE

2.8.1 BASIC commands may be executed from machine code by setting up the full command, as if had been entered from the keyboard, and then jumping to the code that processes a command following a 'RETURN'. The one additional requirement is that the command must be followed by an ":EXEC &HXXXX" to return control to the calling routine. It is not possible to execute some commands in this way as control is automatically passed

to the BASIC command level - the relevant commands detailed in Section 6 include reference to the fact that control is returned to the BASIC command level. The Hitachi assembler (ASM9) uses this method to execute its save and load functions - it executes a SAVEM or LOADM command followed by an EXEC.

2.8.2 The following routine will execute a BASIC command from machine code:-

```
       LDX  #COMMND  Pointer to one byte before start
                     of command
       STX  $BC      BASIC's command line pointer
       JMP  $A450    Execute BASIC command
COMMND FCC  / LINE-(639,199),PSET,6,B:EXEC &H4F3A/
```

2.8.3 The above example has the BASIC command assembled within the machine code program. It is also possible to build up the command from keyboard input as in ASM9. The command may also be set up in the BASIC command line starting at $0278 - location $BC would be set to $0277.

2.9 RECOVERING FROM A "NEW" COMMAND

2.9.1 It is possible to reconstruct a BASIC program following a "NEW" command, provided the BASIC program area has not been overwritten. The "NEW" command resets the "next line" pointer at the start of the first line of the program to zero - this is the same as having no program input. In addition, the "NEW" command resets the pointers to the start of variables, arrays and free space to the same value as the start of BASIC. Therefore, to recover from a "NEW" command it is necessary to reset these pointers.

2.9.2 The steps involved in recovering are as follows:-

1 - Find the address of the start of the BASIC program area from locations \$1D-1E
2 - Find the address of the start of the second line of the program - the first line will be terminated by a zero byte
3 - Replace the first two bytes of the first line (the address found in step 1) with the address found in step 2 - this will recover the "next line" pointer on the first line of the program
4 - Find the end of the BASIC program - follow the chain of "next line" pointers until a pointer points to two zero bytes (there should be three consecutive zero bytes as the last line will be terminated by a zero byte and then two more zero bytes)
5 - Find the address of the byte following the end of the BASIC program - this will be the byte following the three zero bytes found in step 4
6 - Place the address found in step 5 into locations \$1F-20, \$21-22 and \$23-24
7 - The BASIC program is now recovered and should LIST and execute provided the code has not been overwritten.

2.9.3 The above steps may be confusing to follow and it may be easier to experiment with a sample program. Before entering a "NEW" command note the contents of locations \$1F-20 and also the contents of the first two bytes of the program (the "next line" pointer at the start of the first line). These two values are the ones that are calculated in steps 5 and 2 above, respectively.

2.10 RELOCATING BASIC

2.10.1 It is possible to relocate the BASIC program area thus providing a "safe" area for machine code programs - the CLEAR command may also be used to create a "safe" area at the top end of available memory.

2.10.2 Relocating BASIC is similar to recovering from a "NEW" command in that the same pointers must be adjusted - locations \$1D-1E, \$1F-20, \$21-22 and \$23-24. It is not possible to relocate a program that is residing in the BASIC area without resetting all its "next line" pointers. The method outlined assumes that no program is resident and that the relocation will be performed before loading a BASIC program.

2.10.3 The changes required to relocate BASIC are to reset locations \$1D-1E, \$1F-20, \$21-22 and \$23-24 to a new value - all these locations should be set to the same new value. In addition, the byte immediately before the new start of BASIC (one less than the new value) must be set to zero.

3.1 INTRODUCTION

3.1.1 File handling and input/output (I/O) in general appear to be well structured. There are several specialised routines at various places within the ROM which handle specific functions - where relevant these are referred to below or in other sections of the notes. For standard input/output the ROM has a set of routines for each physical device. For each device there is a "jump table" which gives the jump addresses for the standard I/O routines. In most cases I/O may be redirected from one physical device to another by changing the file number in location $9E.

3.1.2 The following notes and table give details on these "jump tables", device handlers and I/O routines.

3.2 NOTES ON PHYSICAL DEVICES

3.2.1 The following notes relate to the standard physicsl I/O devices available. They are intended to represent a brief overview rather than a detailed explanation of the operation of a particular device. In general, the notes relate to the way the devices are handled in the ROM and not the hardware.

3.2.2 KEYBOARD

3.2.2.1 The keyboard circuitry uses hardware scan of the keyboard to determine if a key has been pressed. The scan counter may be obtained by reading location $FFE0. When a key is pressed an IRQ interrupt is generated and the interrupt routine reads location $FFE0. The software routine then converts this to an ASCII character. A list of the $FFE0 values is contained in Appendix D.

3.2.2.2 The software maintains a 32 character keyboard advance buffer which allows keys to be pressed faster than the program is reading them (refer Section 4.4). Further details on handling the keyboard are contained in Section 4.

3.2.3 SCREEN

3.2.3.1 Full details on screen handling and graphics are contained in Section 5. The screen is memory mapped and is configured differently for low resolution and high resolution modes. The standard screen I/O routines determine the resolution and output to the screen accordingly.

3.2.4 CASSETTE

3.2.4.1 The cassette is handled by the ACIA chip and then converted by the hardware to FSK (Frequency Shift Keying) signals before being output to the cassette recorder. Both COMO and CASO use common hardware and may not be used concurrently. Location $01F0 indicates which device is in use. Refer to Section 3.10 for further details on CASO including the file format.

3.2.4.2 The cassette is buffered with the 255 byte buffer area pointed to by location $01D0. Refer to Section 8 for details of a possible conflict between the cassette buffer and the light pen. Section 8 also contains notes on overcoming the "Device In Use" error that sometimes occurs when using CASO and cannot be cleared by closing the relevant file.

3.2.4.3 Most cheap cassette recorders appear to work satisfactorily. The Hitachi brochures recommend a particular Hitachi cassette recorder, however, this appears to be expensive and not readily available. The TANDY CTR-80A cassette recorder works satisfactorily without any apparent problems. One problem with some recorders is that the record lead causes some form of interference. If problems are being encountered with

reading from cassette then try disconnecting the record lead when loading.

3.2.5 RS-232 SERIAL PORT

3.2.5.1 As mentioned above COMO cannot be used concurrently with CASO. Serial input causes an IRQ interrupt, provided the file has been previously opened to COMO, and the input character is stored in the COMO buffer (pointed to by location [01E0]+$1F). COMO has a 128 byte input buffer separate to the cassette buffer. Refer to Section 3.9 for further details on COMO handling.

3.2.6 PARALLEL PORT

3.2.6.1 The parallel port (LPTO) uses one half of the PIA chip to drive a centronics compatible printer. The other half of the PIA is used for switching the additional memory cards. LPTO does not appear to be buffered. The PIA is capable of generating an IRQ interrupt and is connected to the IRQ line, however, the interrupt routine does not appear to handle a PIA interrupt. This is consistent with the relevant code in the ROM ($ED11-ED93).

3.2.7 DISK

3.2.7.1 The I/O handling for the disk drives is handled by the DOS. The transfer to DOS is handled by jumps located in the system work area in low memory - locations $0161-0178 (refer Appendix A). Disk files are distinguished from other files by the top bit of the byte in the open files list (refer Section 3.7).

3.3 DEVICE HANDLER MAP

		KYBD:	SCRN:	CASO:	COMO:	LPTO:	DISK
Jump Table	(X)	F508	FOAB	E20C	OEB1	OF50	--
OPEN	[4,X]	F51A	FOCA	E22A	E918	FOCA	0164
CLOSE	[6,X]	F520	FODO	E2B9	E931	ED11	0167
INPCHR	[8,X]	F7C3	F7C3	E2EC	EA2A	E699	016D
OUTCHR	[10,X]	FOD1	FOD1	E307	E9ED	ED1C	016A
$E84B	[12,X]	FOBD	FOBD	E336	EAF8	EAF8	0170
EOF	[14,X]	F7DF	E699	E21E	EB01	E699	0173
LOF	[16,X]	F7E6	E699	E226	EB10	E699	0173

3.4 STANDARD INPUT/OUTPUT ROUTINES

3.4.1 The above table lists the main routines used for standard input/output. Ther are many other "non-standard" routines within the ROM, especially relating to screen output, and many of these are detailed elsewhere in the notes.

3.4.2 The first line of the above table relates to the "jump table" for the particular device. The offset in square brackets against each I/O routine relates to an offset from the start of the jump table. For example, if the X register contained $F508 then a JSR [8,X] would be a jump to the keyboard character input routine. The JSR [8,X] always relates to character input with the X register determining the relevant device type.

3.4.3 OPEN - [4,X]

3.4.3.1 The OPEN code for KYBD, SCRN and LPTO checks that the file is being opened for input or output as appropriate and has no effect on the physical device. The general OPEN code, not relating to a particular device, sets up the correct value in the open files list (refer Section 3.7).

3.4.3.2 The OPEN code for COMO and CASO sets up the registers and flags for the particular device. For CASO the file header is written to the tape if opened for output - refer Section 3.10 for details of the header format.

3.4.3.3 The code for LPTO allows more than one file to be open at any one time. For COMO only one file may be open for output and one file open for input. The default parameters for COMO are not taken from the switch settings but are hard coded into the ROM - refer Section 3.9.9. The TERM command uses the switch settings for the default values.

3.4.4 CLOSE - [6,X]

3.4.4.1 The CLOSE code for KYBD and SCRN is simply a RTS instruction. For LPTO the code checks if a <CR> or <LF> has been output and outputs them if not. For COMO and CASO the code checks the physical device and closes it down - for output to CASO any characters remaining in the buffer are output before closing. The general CLOSE code, not relating to a particular device, handles the resetting of the open files list (refer Section 3.7).

3.4.5 INPCHR - [8,X]

3.4.5.1 The INPCHR routine is called from within the standard character input I/O routine (also referred to as the $E804 routine) and inputs a character from the relevant device to the A register. The "$E804" refers to the starting address of the general code not relating to a physical device. All input devices are buffered and this routine returns the next character from the buffer.

3.4.6 OUTCHR - [10,X]

3.4.6.1 The OUTCHR routine is called from within the standard character output I/O routine (also referred to as the $E820 routine) and outputs the character contained in the A register. The $E820 refers to the starting address of the general code not relating to a physical device. CASO and disk are the only devices that are buffered on output.

3.4.7 $E84B - [12,X]

3.4.7.1 The precise use of this routine is unclear. It is used for the POS function and also in several other places [B027, B082, B0B5, B0C2, B0CE, B0EF, D108, E094]. The routine returns values into locations $B0-B3. It appaers to return the buffer pointer in some cases (SCRN, COMO and LPTO) but this is not consistent for other devices.

3.4.8 EOF - [14,X]

3.4.8.1 The EOF code returns a "Bad File Mode" error for SCRN and LPTO. For KYBD and COMO it returns a true value if the input buffers are empty. For CASO and disk it returns true if actual end of file has been encountered.

3.4.9 LOF - [16,X]

3.4.9.1 The LOF code returns a "Bad File Mode" error for SCRN and LPTO. For the other devices it returns the number of characters remaining in the input buffer.

3.5 FILE NUMBERS

3.5.1 The BASIC manual refers to file numbers in the range 1 to 16. Some examples of machine code routines infer that in machine code file numbers may range from 0-255. This is not correct as the open files list is structured to cater for file numbers in the range 0 to 17 only. The OPEN code will allow file numbers greater than 17 to be opened, however, it is likely to overwrite a byte in some adjoining area, e.g., the device jump tables or the light pen status flags. This may cause problems when trying to close the file as the byte may have been set to zero and the CLOSE code will assume the file is already closed. Therefore, file numbers should only be in the range 0 to 17.

3.5.2 File numbers 0 and 17($11) are used by BASIC for specific purposes. File number 0 is used by the system for default output to the screen or default input from the keyboard. File number 17($11) is used by the system to access other files without using file numbers 1 to 16, e.g., the LOADM command uses file number $11 to load from cassette or disk. The CLOSE code for BASIC closes all files from 0 to 17 inclusive.

3.6 DEVICE NUMBERS

3.6.1 The standard device numbers are as follows:-

0 - KYBD	$80 - Disk Drive 0
1 - SCRN	$81 - Disk Drive 1
2 - CASO	$82 - Disk Drive 2
3 - COMO	$83 - Disk Drive 3
4 - LPTO	

3.6.2 The relationship between device types (KYBD, SCRN, etc.) and device numbers is detailed in Section 2.8. The device number is stored with the file number in the open files list. The current device being accessed is stored in location $0206. This location is set up by the OPEN code and is determined by the device type preceeding the label, e.g., "CASO:DUMMY" would set location $0206 to 02.

3.6.3 The OPEN code also handles labels without any preceeding device type, e.g., "DUMMY". This label would assume CASO for a cassette based system and Disk Drive 0 for a disk based system. The default device type is contained in location $011B. This is a useful location for running programs that assume, say, Disk Drive 0 in that if Disk Drive 1 is preferred location $011B may be changed to $81 and the program should then run from Drive 1. This will not work for the DSKI, DSKO, etc., commands that specifically include the drive number. Location $011B is only used where the device type or drive number is not specified.

3.7 OPEN FILES LIST

3.7.1 The open files list is pointed to by location $01D4 and is 18 bytes long - refer to Appendix A for exact locations for each mode. The 18 bytes are for file numbers 0 to 17 and the relative position within the list is defined by the file number. The values within the list are as follows:-

Non-Disk Files	- 1X=input 2X=output X=device number
Disk Files	- 9X=input AX=output CX=random X=number of disk buffers remaining available.

3.7.2 For disk files the top bit is set and the bottom four bits indicate the number of disk files remaining that may be opened, i.e., the number of disk file buffers remaining unallocated. The number depends on the maximum number of disk files allowed to be opened at DOS configuration time. The CONFIG program must be run to change this number. As each disk file is opened this number is decremented by one.

3.8 DEVICE LIST

3.8.1 The device list is a list of addresses that points to the "jump table" for the particular device. The jump tables are detailed in Sections 3.3 and 3.4. The device list is pointed to by location $01D2 and the relative position within the table defines the device number. Disk device handlers are not included in this list as all disk I/O is handled by the DOS. The $E402 label processing routine (refer Section 3.10.2) determines disk files by the fact that the device mnemonic is the drive number followed by a colon - if the second character is a colon then the first character is checked for numeric and the top bit is set, e.g., "2:..." generates a device number of $82.

3.8.2 For non-disk files the device mnemonic is checked against the first four characters of each jump table. Each jump table has the relevant device mnemonic as the first four characters. The device mnemonic is differentiated from the label by the colon in the fifth character position - if the colon is not included then the mnemonic may be taken as the label. It is possible to have a file called, say, "CASO" as no colon is included.

3.8.3 Section 3.3 defines the device handler map and the routines listed under "Jump Table" are the addresses in the device jump table that follow the four character device mnemonic. The "Jump Table" line points to the start of each device jump table. The indirect offset is the position from the start of the table, i.e., [4,X] is the first entry after the device mnemonic.

3.8.4 It would be possible to define additional device types by including a pointer in this list. It is assumed that LPT1, COM2, etc. are catered for in this way.

3.9 NOTES ON COMO

3.9.1 As mentioned in Section 3.2.5 COMO and CASO may not be used concurrently and COMO causes an IRQ interrupt on input provided the file has previously been opened to input. The 128 byte COMO buffer is located at location [01E0]+$1F. The COMO jump table and work area is located at [01E0]. Limited details on this work area are contained in Appendix A.

3.9.2 Whilst COMO input causes an IRQ interrupt, it is not possible to have the standard interrupt routine pass control to a machine code routine. The interrupt routine sets the relevant COMx status flag at [01D8], places the character in the input buffer and returns. BASIC checks for this flag when processing each statement ($EB1E - $EB98) and executes the ON COM routine if set. It would be possible to place some code before the normal interrupt routine and test for a COMx interrupt but this would require care to ensure the other IRQ interrupts were not affected (keyboard and light pen). The COMx interrupt routine is located at $EA97 to $EAF7.

3.9.3 One feasible method for detecting COMx input in a machine code routine is to test the relevant COMx status flags each time through the main loop. The pointer for the ON COMx line number could be used to provide a variable jump address for COMx input.

3.9.4 The code in the ROM does not normally allow a LOAD from any device except CASO or disk. There is a RAM jump ($015E) which generates the "Invalid Function Call" error message. It is possible to alter this jump to handle a LOAD from COMx but this is non-trivial as the complete LOAD command would have to be handled by this code.

3.9.5 The COMO input routine detects ASCII SO ($0E) and SI ($0F) characters when 7 bit operation is selected. The SO character selects graphics mode. All characters (except the control characters $00 to $1F) input after that will have the top bit set when they are stored in the buffer. The SI character selects character mode and all characters input after that will have the top bit cleared before they are stored in the buffer. The SO and SI characters are never stored in the buffer for 7 bit input and so are not available to a user program.

3.9.6 The COMO output routine detects the output of graphics characters ($A0 to $FF) for 7 bit operation and selects graphics mode by transmitting the ASCII SO ($0E) character when the first graphics character is output. Character mode is reset when a subsequent ASCII character ($20 to $7F) is output by transmitting the ASCII SI character ($0F). A flag at [$01E0]+$12+$0A indicates which mode is selected.

3.9.7 The TERM command sets a flag ($03D6) to disable the CTRL-S and CTRL-C functions. If CTRL-D is input from the keyboard while a program is using COMO for input or output the program will be aborted. This works for TERM as well as machine language programs.

3.9.8 The TERM command allows direct cursor addressing by the remote device. The cursor address command is the ASCII ESC ($1B) character, followed by '=' followed by a one byte column address, then a one byte line address. The colunm and line address is the hex value of the address byte minus $20. The TERM command does not follow the Teletype convention of suppressing the character after the ASCII ESC ($1B) character (except for direct cursor addressing).

3.9.9 The TERM command uses the switch settings for its default parameters. COMO is different in that it ignores the switch settings and uses values that are preset in the ROM. The default values are stored in location [01E0]+$12+$07 ([01E0]+$19) and this location is preset to $12 which gives default parameters of "S8N2".

3.9.10 The ACIA used in the Peach is a HD46850P. A sample program is include in Section 9.8 which illustrates programming the ACIA. The program implements the Teletype break function for COMO output. The break is generated by the BREAK key, using an interrupt routine, whenever COMO is open for output (as it will be in TERM).

3.9.11 Section 9.7 contains further notes on the RS232 signals and gives details of a loopback plug and crossover cables for the Peach.

3.10 NOTES ON CASO

3.10.1 The CASO input routine verifies each byte written to the buffer, so a memory error in the CASO buffer area will cause an I/O error.

3.10.2 CASO is handled by the ACIA and uses the equivalent of the COMO values "S8N2" together with the hardware switch CS3 position 2 to get 600 bps. The format of a CASO file is set out below:-

FILE FORMAT

Header block
[Data block]
.
.
[Data block]
Trailer block

BLOCK FORMAT

Synchronization data (at least 6 $FF bytes for input; for output see $0119 and $011A in Appendix A)
Header bytes $01, $3C, block type (set $01EE), length (set $01EF)
Optional data (written to buffer)
Checksum (Mod $FF sum of all bytes in header and data)
Desynchronization data (5 $00 bytes output but not read in)

HEADER BLOCK DATA

Label (8 bytes)
File type (from $01FB), File format (from $01FC),
File format (read from $01F4 on output, written to $01F4 and $01F6 on input)
Null characters (9 bytes)

DATA BLOCK DATA

Up to 255 bytes of data

TRAILER BLOCK DATA

No data in trailer block (used to determine end of file).

3.11 FILE HANDLING AND I/O ROUTINES

3.11.1 The following notes relate to standard I/O routines and differ from the notes in Section 3.4 in that these notes do not relate to specific physical devices. Other specialised I/O routines are included in other sections of the notes, especially Section 4 on keyboard handling and Section 5 on screen handling.

3.11.2 LABEL PROCESSING

3.11.2.1 The following routine is an example of file label processing including device mnemonics:-

```
      LDD   #$01FF      File type and format
      STD   $01FB       Store in $01FB and $01FC
      STB   $01F4       Set to same as $01FC
      LDA   FILNUM      File number (1 - 16)
      STA   $9E         File number location
      LDY   $BC         Save location $BC - Y register is
                        not used in $E402 routine
      LEAX  LABEL,PCR   Set pointer to start of label
      STX   $BC         $BC now points to label
      JSR   $E402       Label processing routine
      STY   $BC         Avoids 'Syntax Error' message
      ........          Open file or other processing as
                        required
LABEL FCC   /"CASO:DUMMY"/  Note - quotes must be
                            included - the device type
                            may be omitted.
```


3.11.2.2 The above routine assumes that the actual label is a data field assembled into the code. If setting the label from keyboard input then the above code (LEAX LABEL,PCR) should point to the relevant input buffer area. The opening and closing quotes must be included, either by the machine code routine or by the keyboard input.

3.11.3 OPEN

3.11.3.1 Before opening a file the above label routine, or equivalent, should be executed. The label routine sets up the device type in location $0206 and the file number in location $9E. Notes on executing an OPEN command are contained in Section 6.40.

3.11.4 CLOSE

3.11.4.1 The close routine checks the open files list to check the file has been opened and will not close the file if the relevant byte is set to zero. If for some reason the open files list has been corrupted then the file may not be closed and a "Device In Use" or "File Already Open" message result (refer Section 8 for possible fixes for these messages). Notes on executing a CLOSE command are contained in Section 6.6.

3.11.5 CHRINP - $E804

3.11.5.1 The standard character input routine is executed by a JSR $E804. The relevant file number must be set up in location $9E before this call. File number zero will input from the keyboard and does not need to be opened before the call.

3.11.5.2 For KYBD and COMO the $E804 routine will not return unless a character is in the input buffer. For CASO and disk the routine will return the next character until end-of-file (EOF). The EOF flag is returned in location $9F and is set to $FF on the call after the last character has been returned. The routine will continue to return $FF in location $9F if called after EOF. If EOF is not encountered then location $9F will be zero. The $9F EOF flag is different to the EOF command and will return different values under different conditions (refer Section 6.16 for details on the EOF command).

3.11.6 CHROUT - $E820

3.11.6.1 The standard character output routine is executed by a JSR $E820. The relevant file number must be set up in location $9E before this call. File number zero will output to the screen and does not need to be opened before the call.

3.11.6.2 For output to the screen the special characters less than $20 will not be displayed but the appropriate action corresponding to the special character will be performed. Refer to the notes on the PRINT command (Section 6.48) for details of a routine that will display special characters less than $20 on the screen.

4.1 INTRODUCTION

4.1.1 The following notes relate specifically to the way in which the keyboard is handled. Appendix D lists various keyboard values and character codes.

4.1.2 The keyboard is being continually scanned by the hardware and when a key is pressed the scan value is returned in the $FFE0 I/O register and an IRQ interrupt is generated. This scan value is then translated into an ASCII character by the keyboard interrupt routine in the ROM. The interrupt routine checks if the CTRL, SHIFT, etc., key was also pressed and generates one ASCII character. The keyboard routine in the software can handle up to 10 keys pressed concurrently.

4.1.3 The BREAK key is handled differently in that it does not return a value in $FFE0 but sets a special flag in $FFC8 (refer Section B.5). Refer to Section 4.5 for further details on handling the BREAK key.

4.1.4 Keyboard de-bounce is handled by the hardware. The keyboard scan counter cycles every 16.25 milliseconds and will only look at each key once per cycle. This 16.25ms delay is sufficient to effectively ignore any bouncing of the keys.

4.1.5 There is a significant difference between keyboard input using the standard full screen editor and keyboard input using standard file handling. The standard file handling routines use the 32 character keyboard buffer and will handle a maximum of 32 characters being entered in advance of the program reading the keyboard. When using the full screen editor the keyboard input is displayed on the screen at the cursor location and when the RETURN key is entered the software takes the information from the screen (screen RAM area) and places it into the command line - up to 255 characters. Refer to Section 4.3 for further details on the full screen editor.

4.2 KEYBOARD WORK AREA

[01CA]	Keyboard status byte - following bit values (0=Off 1=On):- 7 - CAPS Lock 6 - SHIFT 5 - K/H(green) 4 - K/H(red) 3 - CTRL 2 - GRAPH 1 - Not Used 0 - Not Used
+$1	Last value input from $FFE0
+$3	Copy of value written to KB register ($FFE0)
+$4	Number of keys pressed (max 10)
+$05-$0E	Value from $FFE0 for each key pressed
+$19	Displacement from [01C8]+$1B for next character
+$1A	Displacement from [01C8]+$1B for first character
+$1B-$3A	Keyboard advance buffer (circular buffer - 32 ch)

4.2.1 As already outlined the keyboard is handled by the IRQ interrupt routine ($F521-$F7A1). Keyboard handling centres around the above keyboard work area. Appendix D gives details of various keyboard values and character codes.

4.2.2 The keyboard interrupt routine reads the value in the keyboard I/O register ($FFE0). This is the hardware scan value at the time the key was pressed or $FF at the time the last key was raised - the raising of a key also causes an IRQ interrupt. The routine translates the $FFE0 value into an ASCII character taking into account if the CTRL, GRAPH, SHIFT, etc., key was pressed at the same time - it may convert two or three "simultaneous" $FFE0 values into one ASCII character.

4.2.3 Location [01CA] (refer to Appendix A for actual addresses for each mode) shows the current keyboard status, e.g., if CAPS LOCK, KATA/HIRA, SHIFT, etc. is currently pressed or permanently set. For CAPS LOCK the "on" value (bit 7=1) indicates that the CAPS LOCK light is off and upper case characters are generated - this is the normal state at power up.

4.2.4 Location [01CA]+$1 is useful in that it gives the last $FFE0 value input - the last key pressed. Care should be taken in using this value as it may give a misleading result if another key, say SHIFT, was also pressed. It is generally reliable for checking specific keys, e.g., the cursor control keys. This value may also be used to differentiate between duplicated keys, e.g., the numeric keys on the numeric key pad and the standard numeric keys.

4.2.5 Location [01CA]+$3 is a copy of the value written to $FFE0 (refer Section B.17 for details) and may be used to determine if the CAPS LOCK light is on. It may also be used to turn the keyboard off (refer Section 4.4).

4.2.6 Locations [01CA]+$19 and +$1A are pointers into the 32 character keyboard buffer. These locations could be used for keyboard input, however, it is safer to use one of the keyboard routines detailed in Section 4.6.

4.2.7 There are several other locations within the keyboard work area and some of these are referred to in Appendix A and the use for others remains unclear. These locations do not appear relevant to the general understanding and handling of the keyboard.

4.3 FULL SCREEN EDITOR

4.3.1 The full screen editor is exceptionally useful and appears to be well implemented on the Peach. Section 4.6 details a routine that uses the full screen editor for input.

4.3.2 When using the full screen editor the keyboard input is displayed on the screen at the cursor location and when the RETURN key is entered the software takes the information from the screen (screen RAM area) and places it into the command line - up to 255 characters. The information is taken from the screen line where the cursor is located and may include lines immediately above or below depending on the screen line map.

4.3.3 The screen line map is located at location [01DE] and indicates if a line has been continued onto the next screen line. This is how BASIC can determine up to 255 characters of input spanning more than one screen line. If characters are entered past column 80 then the screen editor scrolls the screen up or down at that point and continues input on the next line. The screen line map would indicate that the second line was a continuation of the first. Therefore when RETURN is pressed the screen line map will indicate if the line above and/or below is part of the same input statement.

4.3.4 When RETURN is pressed the information is transferred from the screen RAM area to the command line ($0278-0377). The first 255 characters only are transferred and any additional characters are ignored and will not be included with the command statement.

4.4 KEYBOARD ADVANCE BUFFER

4.4.1 The keyboard advance buffer is generally useful in that it allows up to 32 characters to be input while other processing is continuing. The one problem occurs when the information required is the last key pressed, ignoring any intermediate keys stored in the buffer. In many cases the pointers in the keyboard work area could be interrogated and the last key selected. However, if more than 32 keys have been input then the keys after the first 32 are ignored and will not be in the buffer, i.e., the last key pressed is not in the buffer. This may be a problem where the auto repeat has been used, i.e., keys are being placed in the buffer relatively quickly.

4.4.2 There are two basic methods for "turning off" the advance buffer - there does not appear to be a facility for turning off the buffer within the ROM. The first method is to turn off the keyboard when input is not required. This may cause problems as keys pressed whilst the keyboard is "off" will be totally ignored. The keyboard may be turned off by inhibiting the keyboard IRQ interrupt signal (set bit 6 of $FFEO to zero) or by disabling the keyboard scan counter (set bit 3 of $FFEO to one). Both methods appear to have the same result. Care should be taken in turning the keyboard off, especially using a POKE, as once it is off it cannot be turned back on using the keyboard - the power will have to be switched off and on again!

4.4.3 The second method of "turning off" the advance buffer involves checking the last $FFEO value, in location [01CA]+$1, to determine the last key pressed. The keyboard is not disabled and the contents of the buffer will remain. Appendix D list the $FFEO values for each key. Section 4.6 details a routine to convert $FFEO values to ASCII characters, however, it is probably simpler, and just as accurate, to specifically test the last $FFEO value in location [01CA]+$1.

4.4.4 The last $FFEO value may give misleading results if more than one key has been pressed, especially if one was a "control" key, e.g., SHIFT, CTRL, etc. The value in $FFEO will depend on the sequence the keys were pressed and lifted. In addition the keyboard status byte [01CA] will only indicate SHIFT, etc. while they are actually pressed and will be reset when such a key is lifted. Therefore, the last $FFEO value should be used for checking for specific single keys such as cursor control keys, etc. It should be noted that the $FFEO value is different for most duplicated keys, e.g., the numeric keys, but there are a few exceptions - refer Appendix D for details.

4.5 HANDLING BREAK, CTRL C, CTRL D, CTRL S

0244	BREAK/CTRL C/CTRL D flag - $FF=BREAK 03=CTRL C 04=CTRL D
0245	CRTL D flag - 04=CTRL D
0246	CTRL S flag - $13=CTRL S
03D6	CTRL C/S inhibit flag (not BREAK)

4.5.1 The BREAK key is handled in a specific way within the ROM. The system is set up to handle BREAK by a NMI interrupt, however, for some reason it has not been implemented this way. The NMI interrupt is normally inhibited and there is no NMI interrupt routine, excepting a RTI instruction. BREAK also sets a separate I/O register ($FFC8) when pressed. The ROM checks $FFC8 as part of its normal processing and sets flags accordingly. This makes the "turning off" of the BREAK key more difficult. Details of methods for inhibiting BREAK have not been included in the notes as the explanation of the algorithm defeats its purpose. BREAK and CTRL C/D/S do not cause any interruption to a machine code routine unless they are specifically tested for.

4.5.2 The BREAK key does not set location $0244 when pressed and does not store a character in the keyboard buffer. The software sets location $0244 if $FFC8 is set when checking for BREAK, CTRL C, etc. The CTRL C/D/S keys set locations $0244-0246 as part of the keyboard interrupt routine but are not acted upon until specifically tested for.

4.5.2 BREAK and CTRL C have the same function and only differ in the way they are generated. CTRL D is described in the BASIC manual as causing a "soft (hot) start". This is confusing as in most cases there is no apparent difference. CTRL D sets both locations $0244 and $0245 and it is up to the software to take the required action depending on the values in these locations. As far as can be determined CTRL C and CTRL D are treated the same by BASIC and have the same affect as pressing the BREAK key. A JSR $F7F7 will test for the BREAK key and locations $0244-0245 may be tested after the call for the appropriate action.

4.5.3 CTRL S is used for scroll control and is reset by any subsequent keyboard input. It has no effect unless it is specifically tested for. Pressing CTRL S sets location $0246 to $13 and the keyboard input routine resets $0246 unconditionally. A JSR $F814 will test location $0246 and loop continuously if it is non-zero, until a key is pressed. If it is not set, or a key is pressed, then it will return to the calling routine. Pressing the BREAK key will cause a return to BASIC. If a return to BASIC is not appropriate then location $0246 should be tested without jumping to $F814.

4.5.4 Location $03D6 may be used to "turn off" CTRL C and CTRL S and have them passed through as normal ASCII characters without setting locations $0244-0246. If $03D6 is non-zero then CTRL C and CTRL S will not cause a BREAK or stop scrolling but will be placed in the keyboard input buffer as ASCII characters.

4.6 KEYBOARD ROUTINES

4.6.1 CHARACTER INPUT - $E804

4.6.1.1 A JSR $E804 may be used to obtain the next character from the keyboard buffer in the A register. The file number in location $9E should be zero unless a separate file has been opened to KYBD. This routine will wait until a character is input if the buffer is empty. The end-of-file location $9F will not be set and is not useable with keyboard input. The EOF command will return true if the buffer is empty. This jump is best if input is to be redirected to another device other than the keyboard, e.g., COMO as changing $9E is all that is required.

4.6.2 CHARACTER INPUT - $F82C

4.6.2.1 A JSR $F82C will return a character in the A register if there is one in the buffer otherwise it will set the Zero bit in the CC register. This jump is used by BASIC for the INKEY$ function and may be used to test for keyboard input without waiting if none is available. A BEQ after the JSR $F82C will branch if a character was not present.

4.6.3 STORING CHARACTER INTO KEYBOARD BUFFER - $F750

4.6.3.1 A JSR $F750 will store the contents of the A register in the keyboard buffer and is a method of directing keyboard input from within a machine code routine. The U register should be set to point to the keyboard work area before this call (LDU $01CA). If the A register contains $03, $04 or $13 (CTRL C, CTRL D or CTRL S) then locations $0244-0246 will be set and the code will not be stored in the keyboard buffer.

4.6.4 FULL SCREEN EDITOR - $EF55

4.6.4.1 The $EF55 input routine may be used to take advantage of the full screen editor built into the ROM. This routine will display input anywhere on the screen until the 'RETURN' key is pressed. The information will then be taken from the appropriate screen line to BASIC's command line - refer Section 4.3 for further details. The JSR should be followed by a STX $BC to ensure BASIC's command line pointer is set correctly. If the information is required elsewhere then it should be moved from the command line to the desired area - there is no way to have this performed within the $EF55 routine.

4.6.5 CONVERSION TO UPPER CASE - $D355

4.6.5.1 A JSR $D355 will convert the character in the A register to upper case if it is an alphabetic character. Non-alphabetic characters will be ignored.

4.6.6 CONVERSION OF $FFEO VALUE TO ASCII - $F6BC

4.6.6.1 This routine will convert a value from the keyboard scan counter ($FFEO) to an ASCII character. Care must be taken in using this routine as the code tests various keyboard status flags that may affect the result. In addition the routine will not work if there are characters stored in the advance buffer. The routine is provided as it may be useful for some purpose (?). It is safer to use $FFEO values (refer Appendix D) for processing than converting them to ASCII by this routine. The routine uses the SWI interrupt jump as the ROM code is terminated by a RTI instruction:-


```
LDA     #$7E      Code for JMP
STA     $0106     SWI interrupt jump
LDX     #$F6BC    Address of ROM routine
STX     $0107     SWI interrupt address
LDA     FFEOVAL   $FFEO value to be converted
ANDA    #$7F      Mask off top bit
LDU     $01CA     Points to KYBD work area
SWI               Convert value using SWI routine
CLR     $9E       Default file number
JSR     $E804     ASCII character returned in A
                  register
```

4.7 KEYBOARD LIGHTS

4.7.1 The keyboard lights may be controlled by bit settings in $FFEO. It should be noted that controlling the keyboard lights by this method will not affect keyboard input. The following bits in $FFEO will control the lights:-

Bit 2 - 0=SHIFT light on	1=SHIFT light off
Bit 1 - 0=HIRA light off	1=HIRA light on
Bit 0 - 0=KATA light off	1=KATA light on.

4.8 PROGRAMMABLE FUNCTION KEYS

4.8.1 Function keys are up to 15 characters long and are terminated with a zero byte (the 16th byte is zero if all 15 are used for the key). The PF keys may be varied by altering the locations pointed to by location $01CC. It is not possible to set the keys by jumping into the ROM unless the complete BASIC instruction is executed from machine code (refer Section 2.8).

4.8.2 It is possible to define keys longer than 15 characters but it may upset the screen display. If PF1 is set by the above method to 80 characters in length then PF1 through 5 will show the 80 characters in 15 character increments. PF2 through 5 will not work unless by chance the part of the instruction is valid. Another possibility is to relocate the PF key area, altering the $01CC pointer, and making PF10 up to 255 characters in length (the first 15 characters of PF10 will be displayed).

4.8.3 Function keys are controlled by location $023D and can be turned on by setting $023D to non-zero followed by a JSR $FA40. Setting $023D to zero will turn them off but will not clear the bottom line unless a CONSOLE or CLS command is executed or a CTRL Z character (clear to end of screen) is output. CRTL Z may be executed by printing a $1A character on the screen (JSR $E820) with the cursor in the desired position.

5.1 INTRODUCTION

5.1.1 Screen handling and graphics in BASIC are relatively well catered for but are too slow for some requirements. It is possible to perform screen handling and graphics in machine code without using ROM routines, however, the required code is lengthy and complex if various screen modes are catered for. The ROM contains many screen handling routines which automatically cater for any of the possible screen modes. These notes detail how the screen and graphics are handled and give several useful ROM routines.

5.1.2 The full screen editor built into the ROM is detailed in Section 4.3 and a routine to utilise the screen editor features is detailed in Section 4.6.

5.1.3 The screen is handled in hardware by the CRTC chip. This device may be programmed through a machine code routine provided the correct values are used. Further details on the CRT controller are contained in the relevant technical data sheets and some limited details are given in Appendix B. Section B.4.2 gives some suggested values to try and overcome the instability noted with some monitors.

5.2 SCREEN RAM

5.2.1 One of the limiting features of the Peach is the placement of the screen RAM. The screen RAM starts at location $400 and is variable in length depending on the screen mode. The maximum upper limit of screen RAM is $43FF. Section 5.3 details the addresses for each screen line for each mode.

5.2.2 The screen RAM has a separate colour RAM in parallel. This RAM is not directly addressable except through the $FFD8 colour register. The information contained in this 5 bit wide RAM is the same as bits 0

to 4 of the $FFD8 colour register (refer Section B.16). The colour RAM may be enabled and disabled by setting bit 7 of this register. When enabled, the information contained in bits 0 to 4 are written to the colour RAM whenever an address in the range $400 to $4400 is written to. Similarly, when addresses in this range are read the corresponding information is taken from the colour RAM to $FFD8.

5.2.3 Section 3-1-1 of the BASIC manual recommends that machine code programs should not be located in the range $400-$43FF as the screen colour may change under certain circumstances. The problem with characters changing colour and mode is also sometimes noticeable with BASIC programs especially where a large string area has been defined (CLEAR statement). This problem has not been fully investigated, however, it is believed to relate to the use of the colour register $FFD8. Machine code routines using this area should disable the colour RAM when not displaying data on the screen and also ensure $FFD8 is reset to the appropriate values before writing to the screen.

5.2.4 The colour RAM relates to groups of eight horizontal dots. For the normal 8 by 8 dots that make up a character the colour may only be varied for each line of 8 dots, i.e., the same colour is set for all 8 dots of each line. This limits the colour resolution of the screen but also minimises the amount of colour RAM required to hold the colour information.

5.2.5 The organisation of screen RAM appears to use more RAM than is actually required. For example, in 80 column low resolution mode the system reserves the area $400 to $BFF for the screen RAM. However the range $400 to $BCF only is required to display the 80 by 25 lines. The use of the remaining 48 bytes is unclear - they are set to zero every time the screen is cleared.

5.3 SCREEN ADDRESSES BY MODE

5.3.1 The following table indicates the start address and end address for each line of the screen display for 40 and 80 column mode. The addresses are the same for high and low resolution modes except that in high resolution mode there are eight sets of addresses, each offset by 1024 or 2048 depending on the width, for each screen character position - refer to Section 5.4 for an explanation of high resolution screen organisation.

	40 COLUMN MODE		80 COLUMN MODE	
LINE	START	END	START	END
1	0400	0427	0400	044F
2	0428	044F	0450	049F
3	0450	0477	04A0	04EF
4	0478	049F	04F0	053F
5	04A0	04C7	0540	058F
6	04C8	04EF	0590	05DF
7	04F0	0517	05E0	062F
8	0518	053F	0630	067F
9	0540	0567	0680	06CF
10	0568	058F	06D0	071F
11	0590	05B7	0720	076F
12	05B8	05DF	0770	07BF
13	05E0	0607	07C0	080F
14	0608	062F	0810	085F
15	0630	0657	0860	08AF
16	0658	067F	08B0	08FF
17	0680	06A7	0900	094F
18	06A8	06CF	0950	099F
19	06D0	06F7	09A0	09EF
20	06F8	071F	09F0	0A3F
21	0720	0747	0A40	0A8F
22	0748	076F	0A90	0ADF
23	0770	0797	0AE0	0B2F
24	0798	07BF	0B30	0B7F
25	07C0	07E7	0B80	0BCF

5.4 SCREEN ORGANISATION

5.4.1 The screen on the Peach is memory mapped, that is, the information displayed on the screen is mapped into the screen RAM. Changing a byte in screen RAM will change information displayed on the screen. The information displayed on the screen is refreshed directly from screen RAM by the CRT controller chip. This method allows for direct manipulation of the information displayed on the screen.

5.4.2 The screen RAM is organised differently for low and high resolution modes. In low resolution mode there is a direct relationship between a byte in screen RAM and a character position on the screen - one character position occupies one byte in screen RAM. Therefore, in low resolution mode it is possible to write directly to screen RAM and change character positions on the screen (refer Section 5.8).

5.4.3 In high resolution mode it requires eight bytes to represent one character position, both characters and graphics. The eight bytes are offset by 1024 for 40 column mode and 2048 for 80 column mode. For characters it is necessary to write the ASCII value into all eight bytes. For graphics each of the eight bytes represents the desired dot pattern for the particular row of the 8 by 8 matrix. Section 5.8 details the methods for direct screen manipulation from BASIC.

5.5 COLOUR INFORMATION

5.5.1 The colour information for each dot on the screen is stored in the separate 5 bit wide colour RAM. As stated in Section 5.2 the colour resolution of the Peach is limited to groups of 8 bits, that is, each line of 8 bits within the 8 by 8 character matrix. The colour RAM stores the colour and screen attributes for each of these groups of eight dots.

5.5.2 This colour RAM is not directly addressable except through the $FFD8 colour register. Whenever a byte is written into screen RAM the contents of $FFD8 are also written to the corresponding position in colour RAM, unless the colour RAM has been disabled (refer Section 5.2). Similarly, when a screen RAM byte is read the corresponding colour information is placed in $FFD8 by the hardware. BASIC retains a copy of the current $FFD8 value in location $0116.

5.5.3 The background colour is controlled by the $FFD0 I/O register. BASIC retains a copy of the $FFD0 register in location $0249. The background colour may be varied by setting bits 0 - 2 of this register.

5.6 SCREEN PAGES

5.6.1 It is possible to have up to 16 separate screen pages, depending on which mode the system was initialised to - refer to Section 2-1-16 of the BASIC manual. The SCREEN command may be used to switch between screen pages.

5.6.2 The screen page facility allows for multiple screen displays to be set and called when required. It is only possible to set screen pages within the limits of the screen RAM area set up on initialisation. Locations $A3-A4 to $A5-A6 define the screen RAM limits of the current screen page. Location $01C8-01C9 defines the upper limit of screen RAM as set on initialisation.

5.6.3 The SCREEN command may be executed from machine code - refer Section 6.55 for details. It is also possible to move blocks of memory from user RAM to screen RAM provided the colour information can be handled. If the screen information is predominately one colour then the colour register ($FFD8) may be set and followed by a block move. Limited colour variation may be handled by multiple moves with $FFD8 changing before each move. This method is obviously limited as it only

takes a small amount of colour variation to make it unworkable, especially in high resolution mode.

5.7 CONSOLE WINDOWS

5.7.1 Console windows may be set up by executing the CONSOLE command - refer to Section 6.11 for details. Locations $023A-023D may be set up as in Section 6.11 and followed by a JSR $DF6A to set the console window, "Home" the cursor and set the PF keys according to location $023D.

5.8 DIRECT SCREEN RAM MANIPULATION FROM BASIC

5.8.1 Direct screen manipulation from machine code is relatively simple provided the colour register ($FFD8) is set appropriately and high resolution mode is catered for.

5.8.2 In BASIC it is necessary to set up the $FFD8 colour register for graphic or characters and also to set location $0116 (BASIC's copy of last $FFD8 value). BASIC uses location $0116 to set up $FFD8 before any screen manipulation. Under certain conditions the $FFD8 colour register is defaulted back to graphic mode unconditionally without updating $0116. Therefore $FFD8 should be reset before POKEing into screen RAM to ensure the correct character/graphic mode - in many cases one POKE to $FFD8 appears to work satisfactorily.

5.9 SETTING TABS

5.9.1 TABs may be set at any screen column position and may be input from the keyboard using CTRL T to set a tab at the cursor position and CTRL Y to clear a tab at the cursor position.

5.9.2 The TAB information is stored within the ROM at locations $01BE-01C7. These 10 bytes represent the 10 groups of eight columns across the screen starting at column zero. The default value is "01010101010101010101" which indicates a TAB set on every eighth column starting from zero at the left of the screen. This is not a strict bit map or bit representation of columns between 0 and 79. A value of "03" for the first byte would indicate a TAB set at columns 0 and 1 - a value of "83" would indicate TAB settings on columns 0, 1 and 7.

5.9.3 TABs may be set by varying the values in locations $01BE-01C7 as indicated above. Alternatively they may be set or cleared for particular columns by setting the column number in location $0238 followed by a JSR $F260 to set a TAB or JSR $F267 to clear a TAB.

5.10 GRAPHICS HANDLING

5.10.1 Displaying graphic information on the screen involves setting particular bit patterns or specific graphic characters. Individual dots on the screen may be varied by setting the appropriate bits in the screen RAM. This is a lengthy process for complex graphic displays.

5.10.2 The BASIC graphic commands and functions may be executed from machine code as detailed in Section 6. The execution of these routines should assist with complex graphic displays. In some cases it may be faster to move preset bit patterns from user RAM to screen RAM provided the colour information can also be handled.

5.11 SCREEN SCROLLING

5.11.1 Scrolling is normally upwards except where a line is extended to more than one line. In some cases the screen will scroll downwards or parts of the screen will scroll upwards. Scrolling of the screen is controlled by printing specific characters in specific locations.

5.11.2 Downwards scrolling of the screen may be achieved by printing an "Insert" character at the top right hand corner of the screen. This will scroll all lines downward except the top line. In BASIC this may be achieved by a "LOCATE 79,0", or "LOCATE 39,0", followed by "PRINT CHR$(&H12);". The one possible problem with this is that the screen line map will indicate the second line as an extension of the first (refer Section 4.3) and may cause problems if the full screen editor routine is being used.

5.11.3 Various other scrolling effects may be achieved by utilising the "Insert" character in conjunction with the console window. It is possible to "fool" the system by temporarily setting the console window and forcing scrolling by using the "Insert" character or normal scrolling functions. The possible combinations are too numerous to detail but may be experimented with using the full screen editor in conjunction with the LOCATE and CONSOLE commands.

5.11.4 The scrolling code within the ROM appears to be around $F200. Experimentation with jumping into the ROM has produced unpredictable results and it would appear safer to use the standard characters detailed above.

5.12 SCREEN CONTROL CHARACTERS

5.12.1 Section C.2 details jumps for the special characters that have some defined effect on the screen. The relevant function may be achieved by a JSR to the address listed or by outputting the particular character to the screen, using a JSR $E820, at the desired location.

6.1 INTRODUCTION

6.1.1 The following notes relate to specific commands only and are not intended to represent a detailed explanation of all commands. Some commands have not been included as they serve no useful purpose except as part of BASIC programs, e.g., REM, DATA. In other cases, the details on how the command works has not been established to the level that it could be included in this section.

6.1.2 The heading of each command detailed below is followed by an address. This address represents the start of the relevant code as used by BASIC and is the same address as detailed in Appendix C.

6.1.3 Some commands use specific locations as flags, etc., and, where relevant, these are detailed after the heading with the same description as contained in Appendix A.

6.1.4 The following notes give details on executing commands from machine code. The routines and jumps detailed have been tested to reasonable levels and should work under most conditions. It is possible (even likely) that some of the code may not work as expected under some conditions. The notes generally outline the purpose of the code and inspecting the disassembled ROM code around the relevant area may provide the answer to some problems. If the code cannot be made to work then it should be referred to THE SOFTWARE HOUSE for investigation. No guarantee is given that the code will be made to work, however, all reasonable attempts will be made to solve the particular problem.

6.2 AUTO - $B42C

007C	AUTO command flag
007D-007E	AUTO starting line number
007F-0080	AUTO increment value

6.2.1 The code for AUTO edits the command, sets up the parameters, removes the return address from the stack [$B459] and then jumps to BASIC. It is possible to jump into the code with locations $7C and $7D-7E set appropriately and use the default increment of 10 (JSR $B440). It is preferable to set all locations to the desired values and then jump directly to BASIC (JMP $A440). This method avoids any stack problems but relies on the calling routine to set valid parameters.

6.2.2 It should be noted that AUTO will not execute if the starting line number exists and will automatically terminate if an existing line number is encountered.

6.3 BEEP - $F7A2

6.3.1 BEEP allows a number as a parameter but it has no apparent effect. The standard BEEP sound may be obtained by a JSR $F7A7. A JSR $F7A9 will produce the standard keyboard click. Different sounds may be obtained by setting the B register to various values and executing a JSR $F7AB. The standard BEEP uses B=0 and the keyboard click uses B=3.

6.4 CDBL/CINT/CSNG - $CD5E/$CCF7$CD71

0006		Accumulator variable type - 2=int 3=char 4=single 8=double
0056-005D		BASIC's accumulator (3-1-3 of BASIC manual)
005E-005F	??	Negative indicator

6.4.1 These conversion routines will convert a numeric value in one format to another format. The initial value must be correctly set up in BASIC's accumulator, locations $56-5D, with locaton $06 indicating the type. The returned value will also be in BASIC's accumulator with $06 indicating the new type. To execute one of the conversion routines execute a JSR to the relevant address in the heading.

6.5 CLEAR - $DE7C

0025-0026	Top of Stack - lower limit of string space
002B-002C	Top of String space

6.5.1 CLEAR without any parameters resets all variables, strings, arrays, etc. but does not alter either the length of string space or its location. A JSR $A516 will perform this function.

6.5.2 The first parameter of the CLEAR command defines the length of string space. The second parameter defines the top of string space or the top of useable memory as far as BASIC is concerned. This parameter may be used to set up an area of memory that will not be corrupted by BASIC and may safely be used for machine code routines.

6.5.3 The stack is always located at the bottom of string space, i.e., starting at the address contained in location $25. It should be noted that this routine ($A516) resets the stack pointer unconditionally to the address in $25. Therefore, any information stored on the stack before calling this routine will be lost.

6.5.4 To reallocate string space and set the top of memory for BASIC, set up locations $25 and $2B to the desired values and JSR $A516. A JSR will return as normal to the calling routine. A JMP will return to the address defined by the top two locations of the stack. The JMP may be used to set the return address to some location other than the calling routine by pushing the desired address onto the stack before the JMP.

6.5.5 The following code may be executed to take advantage of some of the edits performed by CLEAR but requires the return address to be manually placed on the stack. The code will set up locations $25 and $2B based on the values set up before calling :-

```
LDX     RETADR     Return address
PSHS    X          Store on stack
LDX     STRLEN     Length of string space
PSHS    X          Store on stack - used in edits
LDX     MEMTOP     Top of memory, e.g., $4FFF - use
                   LDX $2B if no change required
JMP     $DE92      Jump to code - JSR will not work.
```

6.6 CLOSE - $E61A

009E	File Number (1-16 for BASIC - 0-17 internally with 0 and 17[$11] reserved for system use)

6.6.1 The CLOSE command closes all file numbers from zero to 17($11). Refer to the notes on the OPEN command (Section 6.40) for details on the use of file numbers outside the range 0 to 17. The following three routines may be executed:-

JSR	$E5F2	- Close file number stored in $9E
JSR	$E5F4	- Close the file number in the B register
JSR	$E628	- Close all file numbers from 0 to 17($11).

6.7 CLS - $F160

00A3-00A4	Start of current screen page
00A5-00A6	End of current screen page
0116	BASIC's copy of colour reg.($FFD8)

6.7.1 A JSR $F160 will clear the current screen page and set the cursor at the top of the console window. The CLS code uses the value stored in $0116 to set up the colour register ($FFD8).

6.7.2 The code sets the address range specified by locations $A3-A4 and $A5-A6 to zeroes. In low resolution mode the values in these locations could be altered to achieve a partial clearing of the screen. This may upset the system unless the correct values are restored immediately after the JSR $F160 and before any scrolling or screen accessing takes place. A safer alternative may be to set the relevant address range to zero without using the CLS code. This alternate method could also be used in high resolution mode by accounting for the way high resolution screens are organised (refer Section 5.2).

6.8 COM(x) ON/OFF/STOP - $EBD8

01D8	Pointer to COM0-COM4 status flags
[01D8]	COM0-COM4 status flags
	+$0 - 00=COM(x) OFF $97=COM(x) ON
	+$1 - $8E=COM(x) STOP
	+$2 - $FF=COM(x) input while 'STOP'
	+$3-4 - Pointer to line for ON COM(x) GOSUB ....

6.8.1 It is possible to jump to the ROM code but requires setting registers to specific values and also more than one jump is required. The simplest way is to set the appropriate bytes as indicated above or to execute the complete BASIC command from the machine code routine (refer Section 2.8).

6.9 CONT - $A5AD

0033-0034	Pointer to current part of BASIC program - used to restart following CONT command

6.9.1 BASIC stores the pointer to the current part of the current line in location $33. The CONT code checks location $33 and gives the error message "Can't Continue" if it is zero. Following a BREAK or STOP any alteration of the program will cause location $33 to be set to zero. A JSR $A5AD will execute the CONT command.

6.10 COLOR - $DFA2

0116	BASIC's copy of colour register ($FFD8)
0249	Copy of Mode Select register ($FFD0)

6.10.1 The COLOR command sets the colour register ($FFD8) and the Mode Select register ($FFD0) as appropriate. BASIC retains a copy of both registers in the above locations.

6.10.2 To alter the color or background color set the above I/O registers as required and adjust locations $0166 and $0249 to match.

6.11 CONSOLE - $DFCF

023A	Bottom line of screen (for cursor back to top)
023B	Console top line number
023C	Console bottom line number
023D	PF Key flag - 00=Key off 01=Key on

6.11.1 The locations above should be set as required. Location $023A defines the line at which the cursor will return to the top line of the console window and should be set to one more than location $023C.

6.11.2 Function keys are controlled by location $023D and can be turned on by setting $023D to non-zero followed by a JSR $FA40. Setting $023D to zero will turn them off but will not clear the bottom line unless a CLS or a CTRL Z is executed. CTRL Z may be executed by printing a $1A character on the screen (JSR $E820) with the cursor in the desired position.

6.11.3 A JSR $DF6A may be executed, after setting the above locations, to "Home" the cursor and set the PF keys according to location $023D.

6.12 DATE$ - $D95F

021C-0221	BASIC's date - one byte per digit

6.12.1 The above locations contain the system date stored one byte per digit. For example "82/10/01" is stored as "08 02 01 00 00 01" in locations $021C to $0221. To alter or set the date the above locations should be set as required.

6.13 DELETE - $B3FA

0081-0082	Starting line no.) LIST/DELETE
0083-0084	Ending line no.) Command

6.13.1 It should be noted that for the following jumps it is not possible to have control returned to the calling routine as it automatically returns to the BASIC command level. To delete all of a BASIC program execute a JSR $B3FF. To delete a selected part of a program set up the starting line number and ending line number in locations $81-82 and $83-84 respectively followed by a JSR $B389 and a JSR $B402.

6.14 EDIT - $F01F

6.14.1 The EDIT command may be executed by setting the line number in location $31-32 followed by JSR $F024.

6.15 END/STOP - $A57B/$A584

008E-008F	Used to store stack pointer while running BASIC
03D3	'Break' flag - +ve='Ready' -ve='Break Ready'

6.15.1 The code for END closes all files and restores the stack from location $8E. STOP is similar except that it does not close files and prints the 'Break' message. There are several possible ways to terminate a BASIC program from a machine code routine or to return to BASIC. To close files execute a JSR $E628 or refer to the notes on the CLOSE command (Section 6.6).

JSR	$A58C	- 'Break' BASIC program and reset the stack
JSR	$A5A0	- use $03D3 to control the printing of 'Break'

```
JSR  $A438  - prints 'Ready' only
JSR  $A440  - returns to BASIC without printing
              'Ready'
```

6.16 EOF/LOF - $E89B/$E89E

009F EOF flag for $E804 input routine - $FF =EOF

6.16.1 The EOF code may be used to check for end of file for a particular file number. Location $9F will also indicate EOF but should only be used immediately after a JSR $E804. The EOF code does not return the correct value for disk files under SSSD Disk Basic - it returns $00FF instead of $FFFF in locations $58-59. The DOSMOD program detailed in Appendix F fixes this problem for SSSD Disk Basic.

6.16.2 The following code will execute the EOF and LOF functions with the result returned in locations $58-59. For LOF the returned value is the number of characters in the input buffer and for EOF a $FFFF value indicates end of file and zero indicates not end of file:-

```
LDA  EOFLOF    #$0E for EOF and #$10 for LOF
LDB  FILENO    File number
JSR  $E8A4     Execute EOF or LOF code
```

6.17 ERROR - $BB02

6.17.1 ERROR is used to simulate BASIC error messages by causing an error condition. The error number should be set in the B register and followed by JMP $A3A7. It should be noted that control is passed to BASIC and cannot be regained by the calling routine.

6.18 EXEC - $DE70

0148 Go address from LOADM or EXEC

6.18.1 The EXEC code simply executes a JMP [$0148]. If a parameter is given then the address is stored in location $0148 otherwise the existing value in $0148 is used. LOADM also sets location $0148 to the start address given with the original SAVEM.

6.19 FILES - $E39E

0161 DOS jump for 'FILES'
0206 Device no. 0=KYBD: 1=SCRN: 2=CASO: 3=COMO: 4=LPTO: $80=0: $81=1: $82=2: $83=3:

6.19.1 Set location $0206 to the desired device type (CASO or Disk) followed by a JSR $E3A1. Note that the cassette FILES statement requires 'ABORT' to be pressed to terminate and will not return to the calling routine.

6.20 HEX$/OCT$ - $B25F/$B260

6.20.1 The HEX$ and OCT$ code will return a string starting at location $037A. To print a hex value on the screen refer to the notes on the PRINT command (Section 6.48). The following routine will execute HEX$ or OCT$ but requires the return address to be placed on the stack before calling:-

```
      LDX  RETADR    Set up return address
      PSHS X         Place on stack
      LDX  VALUE     Value to be converted
      STX  $58       Store in accumulator
      LDA  #$02      Variable type=integer
      STA  $06
      JSR  $B25F     Execute HEX$ code - JSR is necessary
                     but will return to RETADR and not to
                     next location after JSR
   OR JSR  $B260     Execute OCT$ code
```

6.21 INKEY$ - $F868

6.21.1 The INKEY$ function returns a character if present otherwise it returns a null string. INKEY$ uses the $F82C character input routine - refer Section 4.6 for details.

6.22 INPUT - $D06C

009E	File Number (1-16 for BASIC - 0-17 internally with 0 and 17[$11] reserved for system use)

6.22.1 There are several variations to the INPUT statement and these are processed by the routine starting at $CFDA (INPUT "..., INPUT #..., INPUT x). The various formats may be executed by printing the prompt message as required (JSR $B114 will print the '? ' prompt) and then using a JSR $EF55 for the input (refer Section 4.6 for further details on the $EF55 input routine). Location $9E may be set accordingly for input other than the keyboard. Input will be terminated by a <CR>, end-of-file or 255 characters. Characters less than $20 will be ignored but the special characters greater than $20 will be accepted. The input string is assembled starting at location $0278 (the command line) and must be moved separately to another location if

required. Conversion to numeric values must also be handled separately.

6.22.2 The input statement may also be emulated by using the $E804 character input routine and specifically testing for terminating characters, etc.

6.23 INPUT$ Function - $E8DD

6.23.1 The INPUT$ code loops around a JSR $E804 for the required number of characters. The routine requires adjustment of the stack to call from machine code (similar to HEX$). The following code is similar to the ROM routine:-

```
      LDX  POINTR    Points to start of string
      LDB  LEN       No. of characters plus 1
LP    DECB
      BEQ  .....     Jump after LEN characters
      JSR  $E804     Character input
      STA  X,+
      BRA  LP
```

6.24 KEY x,"...." - $FA0D

01CC-01CD	Pointer to start of PF keys
[01CC]	PF Keys

6.24.1 Function Keys are up to 15 characters long and terminated with a zero byte(the 16th byte is zero if all 15 are used for the key). The PF keys may be varied by altering the locations pointed to by location $01CC. It is not feasible to jump into the ROM code unless the complete BASIC instruction is executed from the machine code routine (refer Section 2.8).

6.24.2 It is possible to define keys longer than 15 characters but it may upset the screen display. If PF1 is set by the above method to 80 characters in length then PF1 through 5 will show the 80 characters in 15 character increments. PF2 through 5 will not work unless by chance the part of the instruction is valid. Another possibility is to relocate the PF key area, altering the $01CC pointer, and making PF10 up to 255 characters in length (the first 15 characters of PF10 will be displayed).

6.25 KEY LIST - $F9D5

01CC-01CD	Pointer to start of PF keys

6.25.1 KEY LIST may be executed by a JSR $F9D7. The code uses the address in $01CC to print the details. The special control characters are printed on the screen by a JSR $FA73 with the length of the string in the B register and the address of the string to be printed in the X register.

6.26 KEY(x) ON/OFF/STOP - $FA03

01DA-01DB	Pointer to PF Key status flags
[01DA]	PF Key status flags - 5 bytes for each PF Key +$0 - 00=KEY x OFF $97=KEY x ON +$1 - $8E=KEY x STOP +$2 - $FF=KEY x pressed while 'STOP' +$3-4 - Pointer to line for ON KEY (x) GOSUB ....

6.26.1 This command uses the exact same code as COM(x) ON/OFF/STOP after setting up the relevant PF Key pointer in the X register. Refer to the notes for COM(x) (Section 6.8) for further details on the status flags. Refer to Section 6.11.2 for details on displaying the PF keys on the screen.

6 - NOTES ON SPECIFIC COMMANDS

6.27 LINE - $D67F

00A1	LINE option 00=PRESET 01=PSET $C8=Char. option
00A8-00A9	Starting horizontal position - (Various screen commands)
00AA-00AB	Starting vertical position
00AC-00AD	Ending horizontal position
00AE-00AF	Ending vertical position
03DA-03DB	Last horizontal point from LINE Command
03DC-03DD	Last vertical point from LINE command
03EA	LINE/PAINT/PSET color option - 1X=graphic 0X=character X=color
03EB	Character for LINE command char. marking option

6.27.1 The LINE command sets up all the above locations depending on the options specified. Locations $03DA and $03DC are set up while editing the command and are not used in the following jumps. They may be used to store the ending co-ordinate for use by subsequent code, i.e., set locations $A8 and $AA from locations $03DA and $03DC and then execute one of the following jumps. Locations $A8 through $AF are altered by the following jumps and cannot be used for subsequent jumps without being re-initialised. Location $A1 defines the PSET, PRESET or character marking options and location $03EA defines the color option. The relevant locations above must be initialised before executing one of the following jumps:-

LINE	- JSR $D709 followed by JSR $D79A
LINE ....,B	- JSR $D709 followed by JSR $D6E2
LINE ....,BF	- JSR $D709 followed by JSR $D71C.

6.28 LINE INPUT - $CFBA

6.28.1 The notes on the INPUT command cover the LINE INPUT command (refer Section 6.22). LINE INPUT uses the $EF55 character input routine.

6.29 LIST - $E4BE

009E	File Number (1-16 for BASIC - 0-17 internally with 0 and 17[$11] reserved for system use)
0081-0082	Starting line no.) LIST/DELETE
0083-0084	Ending line no.) Command

6.29.1 It is not possible to execute the following jumps and have control returned to the calling routine as the LIST code automatically returns to the the BASIC command level. To list the complete BASIC program execute a JSR $E4E0. To list a line number range the starting and ending line numbers should be set up in locations $81-82 and $83-84 respectively and followed by a JSR $B389 and a JSR $E4E3.

6.29.2 LIST "...." is exactly the same as SAVE "....",A, except that LIST allows a line number range, and may be executed by either of the above jumps provided the output file is set up beforehand. The file parameters and label should be set up as for the OPEN command (refer Section 6.40). Locations $9E, $01F4, $01FC and $01FB should be initialised but the file should not be opened as it is handled by the $E4E0 or $E4E3 routine. LIST creates an ASCII format file so locations $01F4 and $01FC should be set to $FF. The file type ($01FB) may be any value except 2 (machine code).

6.30 LOAD/LOADM/LOAD? - $E6D4

009E	File Number (1-16 for BASIC - 0-17 internally with 0 and 17[$11] reserved for system use)
020E	02=RUN 03='R' option of LOAD/LOADM
020F	MERGE indicator $FF=MERGE
0210	LOADM indicator 00=LOADM XX=LOAD/MERGE/RUN
0211-0212	LOADM offset value

6.30.1 It should be noted that control is not returned to the calling routine as the LOAD code automatically returns to the BASIC command level. LOAD uses file number $11 and location $9E is set up by the $E717 routine. To execute 'LOAD', 'LOAD ... ,R', 'LOADM', 'LOADM ... ,R' or 'MERGE' the above locations should be initialised appropriately and the following code executed:-

```
      LDY     $BC        Save location $BC - Y register
                         is not used in $E402 routine
      LEAX    LABEL,PCR  Set X to point to label
      STX     $BC        $BC now points to label
      JSR     $E402      Process label according to
                         device type
      STY     $BC        Avoids 'Syntax Error' message
      JSR     $E717      Execute LOAD code
LABEL FCC     /"CASO:DUMMY"/  Note - quotes must be
                              included - the device type
                              may be omitted.
```

6.30.2 The LOAD? command is identical in function to SKIPF. Refer to the notes on SKIPF for details (Section 6.57).

6.31 LOCATE - $E04F

0115	Cursor flag - 00=off $20=static $40=on fast $60=on normal
0238	Screen column no. (0 - 79)
0239	Screen line no. (0 - 24)
00A8-00A9	Starting horizontal position - (Various screen commands)
00AA-00AB	Starting vertical position

6.31.1 To set the position for output of the next character without actually moving the cursor first, set locations $0238 and $0239 as required. This will not move the cursor until the next character is output to the screen.

6.31.2 To relocate the cursor without outputting a character set the desired horizontal position in the A register and the line number in the B register and execute a JSR $F13C.

6.31.3 The cursor itself may be controlled by setting location $0115 as required (values in $0115 are the same as the LOCATE command cursor values multiplied by $20) followed by a JSR $F2AA. The cursor may be turned off by a JSR $F2AE.

6.32 LOF - $E89E

6.32.1 The LOF command is detailed with the EOF command - refer Section 6.16.

6.33 MERGE - $E6CC

6.33.1 The notes on LOAD also cover the MERGE command (refer Section 6.30).

6.34 MON - $DCE6

6.34.1 The MON command may be executed by a JSR $DCE6 but control cannot be regained by the calling routine as MON returns to the BASIC command level following BREAK or CTRL D. MON uses the $EF55 routine for all input (refer Section 4.6 for further details on the $EF55 input routine).

6.35 MOTOR - $EE3C

01F7	Motor - 0=off $FF=on(MOTOR) $80=on(load/save)

6.35.1 MOTOR ON may be executed by a JSR $EE64. MOTOR OFF may be executed by a JSR $EE67. The toggle effect may be achieved by executing a ORCC #$04 followed by JSR $EE3C.

6.35.2 The cassette motor may also be controlled by setting bit 7 of the Motor register ($FFD2). Setting the top bit turns the relay on and clearing it turns it off. Location $01F7 should be set to the same value as stored in $FFD2.

6.36 NEW - $A4FE

6.36.1 The NEW command may be executed by a JSR $A4FE and control will be returned to the calling routine.

6.37 NEW ON - $A4F0

0248	Switch settings (NEW ON value)

6.37.1 The NEW ON command may be executed by a JSR $FAB3 with the normal NEW ON value in the A register.

6.37.2 To initialise the system to the default settings it is necessary to execute a NEW ON with the correct value as a parameter or to switch the machine off and on again. An alternate method is to EXEC &HFAB0 or execute a JSR $FAB0. The $FAB0 routine picks up the default settings from location $0248.

6.37.3 Protection from using the ABORT switch may be achieved by altering any of the reset vector bytes at $FD-FF (zero in any of the three bytes is sufficient). If the ABORT key is pressed then a power on restart will be executed and the system, including all memory, will be re-initialised.

6.38 OCT$ - $B260

6.38.1 The OCT$ command is detailed with the HEX$ command - refer Section 6.20

6.39 ON PEN - $E1BF

01CE	Pointer to L/Pen screen area co-ords.
[01CE]	PEN screen area (1-9) co-ordinates
	+$0 - Starting horizontal position
	+$1 - Starting vertical position
	+$2 - Ending horizontal position
	+$3 - Ending vertical position
	+$4-5 - Pointer to line for ON PEN
+$30-$31	Pointer to line for undefined area

6.39.1 The BASIC manual (Section 2.10.2) states that PEN areas may range from 1 through 9. The PEN statement will allow areas between 1 and 9 to be defined. Use of PEN 9 conflicts with the cassette buffer and using both concurrently will cause problems. The ON PEN statement will allow nine line numbers in total with the first being for the undefined area, i.e., the ON PEN statement only caters for eight defined areas. Use of the ON PEN statement with nine parameters will overwrite some of the PEN 9 details. Therefore, it is suggested that PEN 9 should not be used - PEN 1 to 8 are handled correctly and will not conflict with the cassette buffer or the ON PEN statement.

6.39.2 The ON PEN statement sets up the appropriate line number pointer in the above locations (the PEN number defines the appropriate group of 6 bytes). The first parameter is the line number for the undefined area of the screen and is stored at the end of the above area ([01CE]+$30-$31).

6.40 OPEN - $E64E

009E	File Number (1-16 for BASIC - 0-17 internally with 0 and 17[$11] reserved for system use)
01F4	Used for open - set to same as $01FC (File format) and placed in CAS0 header block
01FB	File Type 0=BASIC 1=Data 2=Machine code
01FC	File format 0=Binary $FF=ASCII

6.40.1 File numbers greater than 17 should not be used as it is not handled correctly by the OPEN code. It appears from the code that only file numbers between 0 and 17 have been catered for. The OPEN code will calculate the relative position in the open files list to store the file status flag, however, for file numbers greater than 17 this relative position will be outside

the defined open files list and may upset the device jump tables, Light Pen status flags, etc. (refer Section 3.5).

6.40.2 Locations $9E, $01F4, $01FB and $01FC should be set as appropriate. Locations $01F0, $01FA, etc., are set by the OPEN code. To OPEN a file execute the following code:-

```
        Set up locations $9E,  $01F4,   $01FC,  $01FB  as
                             appropriate
        LDY     $BC          Save  location  $BC - Y register
                             is not used in $E402 routine
        LEAX    LABEL,PCR    Set pointer to label
        STX     $BC          $BC now points to label
        JSR     $E402        Process label
        STY     $BC          Avoids 'Syntax Error' message
        JSR     $E681        Open for input
     OR JSR     $E684        Open for output
  LABEL FCC     /"CASO:DUMMY"/  Note   -  quotes  must  be
                                included - the device type
                                may be omitted.
```

6.41 PAINT - $DB74

00A8-00A9	Starting horizontal position - (Various screen commands)
00AA-00AB	Starting vertical position
03EA	LINE/PAINT/PSET color option - 1X=graphic 0X=character X=color
03EC	Boundary clolr for PAINT command

6.41.1 To execute the PAINT command the desired co-ordinates should be set up in locations $A8-A9 and $AA-AB followed by a JSR $D554. This jump converts the co-ordinates to screen locations depending on the mode but does not check the co-ordinates for valid ranges. Locations $03EA and $03EC should be set as required followed by a JSR $DB93 to execute the PAINT code.

6.42 PEN x - $E16D

6.42.1 PEN x allows for the definition of areas on the screen and associated processing depending in which area a Pen strike occurs. The BASIC manual states the PEN areas may be between 1 to 9. PEN 9 conflicts with other areas and is not recommended (refer Section 6.39 for details). The PEN area co-ordinates are set up in the same area as for the ON PEN statement and are detailed in Section 6.39.

6.43 PEN Function - $E09F

6.43.1 Refer to Section 2-13-28 of the BASIC manual for details of the PEN function. To execute the PEN function set the B register to the function number followed by a JSR $E0A2. The returned value will be in locations $58-59.

6.44 PEN ON/OFF/STOP - $E160

```
01D6            Pointer to Light Pen status flags
[01D6]          Light Pen status flags
                   $0 - 00=PEN OFF  $97=PEN ON
                   $1 - $8E=PEN STOP
                   $2 - $FF=Pen strike while 'STOP'
              ??  $3-4 - set to the same value as $01D6 ?
```

6.44.1 This command uses the exact same code as COM(x) ON/OFF/STOP. Refer to the COM(x) notes for details (Section 6.8). Bytes 4-5 differ to COM(x) and KEY(x) in that the ON PEN line pointer is stored with the relevant screen area (refer Section 6.39) and not with the status flags. This is obviously necessary as ON PEN can define up to nine different line number pointers and relates to the defined screen area.

6.45 POINT - $D577

00A8-00A9	Starting horizontal position - (Various screen commands)
00AA-00AB	Starting vertical position

6.45.1 The POINT function returns a value to indicate if a dot is present or not at the specified location. The required screen co-ordinates should be set up in locations $A8-A9 and $AA-AB in the same format as in the BASIC manual (Section 2-13-29) followed by a JSR $D57A. The value will be returned in locations $58-59 with $FFFF indicating a dot and zero indicating no dot.

6.46 POS - $E082

6.46.1 The POS function returns the cursor and printer head positions depending on the parameter. A parameter of zero indicates the cursor while 1-16 indicates the printer head position relating to the same file number.

6.46.2 To execute the POS function set up the appropriate parameter value in the B register followed by a JSR $E085. The value is returned in locations $58-59. The POS function uses the $E84B routine to obtain the appropraite value - the exact use of this routine is unclear.

6.47 PRESET/PSET - $D599/D59A

00A1	LINE option 00=PRESET 01=PSET $C8=Char. option
00A8-00A9	Starting horizontal position - (Various screen commands)
00AA-00AB	Starting vertical position
03EA	LINE/PAINT/PSET color option - 1X=graphic 0X=character X=color

6.47.1 The PRESET or PSET command may be executed by setting the appropriate screen co-ordinates in locations $A8-A9 and $AA-AB followed by a JSR $D554. This routine converts the co-ordinates to screen positions depending on the mode but does not edit the initial values for valid ranges. Locations $A1 and $03EA should be set as appropriate and followed by a JSR $D5B4 to execute the PRESET/PSET code.

6.48 PRINT - $B043

009E	File Number (1-16 for BASIC - 0-17 internally with 0 and 17[$11] reserved for system use)

6.48.1 The PRINT code is difficult to follow and specifically tests for functions such as "TAB(", "SPC(", ",", ";", etc. The PRINT command may be executed from machine code by executing the complete command (refer Section 2.8). It is generally easier to use the following jumps to achieve the same effect as the print command. Location $9E should be set as appropriate and the file opened before executing on of the following jumps. For output to the screen file number zero may be used without the need to open the file, or any other file number between 1 and 16 provided it is opened to "SCRN".

JSR $E820 -	A reg.=character to be output $9E=file number
JSR $B104 -	X=one byte before start of string - this routine terminates on a zero byte or '"'
JSR $B10A -	X=start of string B=length - will print all characters up to 'length' - special characters will not print on the screen but will cause the associated function to be executed
JSR $E5A5 -	D=two bytes to be printed (A followed by B)
JSR $B0BE -	Prints 'CR' followed by 'LF'

JSR $B116 - Prints one blank character
JSR $B114 - Prints '? ' prompt
JSR $B119 - Prints '?' prompt (no blank)
JSR $C682 - D=integer value to be printed in ASCII
JSR $DD2C - D=integer value to be printed in Hex - four ASCII characters with zero fill to the left
JSR $DD32 - A=value to be printed in Hex - two ASCII characters with a leading zero if required
JSR $A42C - JSR $B104 followed by return to BASIC. Used for BASIC error messages and control will not return to the calling routine
JSR $FA73 - X=start of string B=length - prints to screen only and will print special control characters without executing appropriate function - used to display PF Keys on screen
JSR $F115 - Prints the character in the A register at the cursor location - ignores the function of the special screen control characters and may be used to print characters less than $20

6.49 <u>RENUM - $BD6C</u>

6.49.1 The RENUM command uses various general purpose locations to store its parameters. Set up the new line number in locations $5A-5B, the increment in $5C-5D and the old line number in $65-66 followed by a JSR $BDAA. Control is returned to the BASIC command level and cannot be regained by the calling routine.

6.50 <u>RESTORE - $BD63</u>

0039-003A Pointer for next DATA statement

6.50.1 To execute a RESTORE to the start of the program perform a JSR $A574. To execute a RESTORE to a specific line number set up the line number in locations $31-32 followed by a JSR $A56C.

6.51 <u>RND - $CC73</u>

0253-0259 ?? Random number area

6.51.1 The RND function and RANDOMIZE uses the above area for various values the exact purpose of which is unclear. The random seed value is stored in locations 0254-0255 and is always set on initialisation to $4FC7. To execute the RND function set a new seed in 0254-0255 if required followed by a JSR $CC70. The random number is returned as a single precision real number in locations $56-59.

6.51.2 A alternate method of obtaining a random number in machine code may be to multiply the keyboard register ($FFE0) by the 60Hz timer counter ($0215). Both these locations are continuous counters and should give a reasonable random value.

6.52 <u>RUN - $A5BD</u>

6.52.1 The RUN statement closes all files before running the BASIC program. To close all files execute a JSR $E628. To RUN the BASIC program in memory close all files if required and the execute a JSR $A510 followed by a JSR $DE3E and a JMP $D3BA.

6.52.2 To RUN a file other than the current program in memory refer to the notes on the LOAD/LOADM/LOAD? commands (Section 6.30) as these notes cover the 'LOAD ".....",R' command which is the same as 'RUN"....."' except that files are not closed by the LOAD code.

6.53 SAVE - $E4FA

6.53.1 To execute a binary save (non ASCII) location $9E should be set as appropriate, the label should be set up as for the OPEN command (Section 6.40) followed by a JSR $E579. Location $9E should be initialised but the file should not be opened.

6.53.2 An ASCII Save (SAVE "....",A) is identical to a LIST "....", except that LIST may have a line number range, and both commands use the same code. Refer to the notes on the LIST command (Section 6.29) for further details.

6.54 SAVEM - $E5AD

6.54.1 To execute a SAVEM locations $9E, $01F4, $01FC and the label should be initialised as for the OPEN command (refer Section 6.40) and the following code executed. Note that the file should not be opened before executing this code. The return address must be manually placed on the stack as the routine uses the stack to store the SAVEM parameters:-

```
Set up label and location $9E
CLR     $01FC     Binary format
CLR     $01F4     Set to same value as $01FC
LDX     RETADR    Return address
PSHS    X         Store on stack
LDX     STADR     Start address - first parameter
PSHS    X         Store on stack
LDX     ENDADR    End address - second parameter
PSHS    X         Store on stack
```



```
LDX     GOADR     Go address - third parameter
PSHS    X         Store on stack
JMP     $E5C2     Jump to code - JSR will not work
```

6.55 SCREEN Statement - $DEB2

```
00A7              0=40 col. low res.   1=80 col. low res
                  2=40 col. high res.  3=80 col. high res.
```

6.55.1 To execute the SCREEN statement the return address and the parameters must be set up on the stack before jumping to the code. There are two jumps, one for changing the interlace mode and one for leaving it unchanged. Both jumps require the same parameters to be set up beforehand:-

```
   LDX     RETADR    Return address
   PSHS    X         Store on stack
   LDA     GRAPHIC   Graphic parameter - YX
                             Y=0  X=$A7 value  - equivalent
                                  to not specifying first
                                  param.
                             Y=8  X=appropriate $A7 value -
                                  do not change width. Y=8
                                  causes screen to be
                                  cleared
   PSHS    A         Store on stack
   LDA     PAGENO    Screen page number less one - 0-15
                     not 1-16 - 0=current page
   PSHS    A         Store on stack
   JMP     $DF09     Interlace mode unchanged - JSR
                     will not work
OR LDB     INTLCE    0=Interlace  1=Non-interlace
   JMP     $DEEB     Change interlace mode - JSR will
                     not work
```

6.56 SCREEN Function - $D840

00A8-00A9	Starting horizontal position - (Various screen commands)
00AA-00AB	Starting vertical position

6.56.1 The SCREEN function may be executed by setting the screen co-ordinates in locations $A8-A9 and $AA-AB followed by a JSR $D638 and a JSR $DE40. This will set up the screen co-ordinate parameters. The select switch (refer BASIC Manual Section 2-13-33) should then be loaded into the B register and followed by a JSR $D857. The character code or attribute will be returned in locations $58-59.

6.57 SKIPF - $E2D1

6.57.1 SKIPF and LOAD? are identical in function and are executed by the same code. To execute SKIPF the following code should be executed:-

```
      LDY    $BC        Save location $BC - Y register
                        is not used in $E402 routine
      LDX    LABEL,PCR  Set X to point to label
      STX    $BC        $BC now points to label
      JSR    $E2D1      Process label and execute SKIPF
      STX    $BC        Avoids 'Syntax Error' message
LABEL FCC    /"DUMMY"/  Note - quotes must be included
```

6.58 STOP - $A5A4

6.58.1 The notes on the END statement also cover the STOP statement. Refer Section 6.15 for further details.

6.59 STR$ Function - $ACF2

6.59.1 The STR$ function converts a signed binary value to a string. The binary value should be set up in BASIC's accumulator (locations $56-5D) and the variable type in location $06. The function is executed by a JSR $ACF7 with the string returned starting at location $037C. The string is terminated with a zero byte and has leading blanks. The $B104 print routine may be used to print the string (refer Section 6.48).

6.60 TERM - $EC49

0207-020A	COMO options, e.g., "S8N2"
03D7	TERM duplex flag - 00=Full 02=Half

6.60.1 The TERM code closes all files and then opens file number 1 for output and file number 2 for input to COMO. To execute the TERM command set up the COMO parameters in $0207-020A and the duplex flag in $03D7 as required followed by a JSR $EC8C. Control will not be returned to the calling routine as the TERM code returns to the BASIC command level on ABORT or CTRL D.

6.61 TIME$ - $D939

0215	60 Hz timer counter
0216-021B	BASIC's time - one byte per digit

6.61.1 BASIC stores the current time in locations $0216-021B one digit per byte. For example "11:59:38" is stored as "01 01 05 09 03 08". Location $0215 is decremented by the timer interrupt routine and BASIC's time is incremented every 60 interrupts. The above locations should be set appropriately to set or alter the time.

6.62 TROFF/TRON - $B45E

0088 BASIC trace flag 0=TROFF $4F=TRON

6.62.1 The code for TROFF/TRON simply sets location $88 as appropriate and returns.

6.63 VAL - $AF8F

6.63.1 The VAL code will convert an input string to a single precision real number. The number may be converted to an integer by executing a CINT command (refer Section 6.4). The VAL code requires location $BC to point to one before the start of the string with the string terminated by a zero byte. To execute the VAL code set $BC to the location of the string followed by a JSR $C5A1. The number will be returned in BASIC's accumulator (locations $56-5D).

6.64 WAIT - $AFF3

WAIT (<address>, <bit mask 1> [, <bit mask 2>])

6.64.1 The WAIT command is not referenced in the BASIC manual. The format is as detailed above. The address is any memory address between 0 and $FFFF. Bit mask 1 is a mask to select a particular bit or number of bits of the address. The command waits until the particular bit is set and then continues. Bit mask 2 may be used to test for bits being unset rather than set. The code performs an exclusive OR on the memory location and bit mask 2 followed by an AND with bit mask 1. The code loops continuously until the result is non-zero.

6.65 WIDTH - $E012

6.65.1 The WIDTH command re-initialises the CRTC register. To execute set the B register to $28 for 40 column and $50 for 80 column followed by a JSR $E015.

7.1 INTRODUCTION

7.1.1 As outlined in Section 2.5 the ROM handles BASIC commands and functions by having tables of commands with corresponding tables of addresses for the code that handles the command. The ROM is structured to handle more than one set of these command and function tables. The Disk Basic code uses the second set of pointers to command and function tables and the third set is available for additional commands.

7.1.2 The method for implementing new commands and functions outlined in this section allows for the full integration of the new commands with the existing commands. The new commands may be executed as direct statements or from within BASIC programs. The code handling the new command may be written to include the use of variables, as well as literals, for parameters.

7.2 COMMAND AND FUNCTION TABLES

0120	Number of entries in BASIC command table
0121-0122	Pointer to BASIC command table
0123-0124	Pointer to addresses matching command table
0125	Number of entries in BASIC function table
0126-0127	Pointer to BASIC function table (tokens = 'FF XX')
0128-0129	Pointer to addresses matching above table
012A	Number of Disk Basic Commands
012B-012C	Pointer to Disk Basic command table
012D-012E	Pointer to Disk Basic command handler
012F	Number of Disk Basic functions
0130-0131	Pointer to Disk Basic function table
0132-0133	Pointer to Disk Basic function handler
0134-013D	Locations for additional command or function tables

7.2.1 The above pointers define the command and function tables implemented for standard BASIC and for Disk BASIC. Locations $0134-013D may be used for additional tables.

7.2.2 For standard cassette BASIC the above pointers point to lists of commands and addresses. The ROM is specifically coded to handle these tables and does not expect them to be varied. For Disk BASIC the pointers point to lists of commands but do not point to lists of addresses - the second pointer points to a command "handler" routine. This is presumably to allow for greater flexibility in that the command handler routine may then vary the way the commands are processed. These command handler routines are located within the Disk BASIC code.

7.2.3 The third set of pointers, locations $0134-013D, are not normally defined except that the pointer to the "handler" routine points to code that generates a "Syntax Error" message. The code within Disk BASIC jumps to this address if a command does not match its tables. Therefore, if a new command is implemented then this third set of pointers may be used to handle the command before a "Syntax Error" message is generated.

7.2.4 The concept of BASIC tokens is outlined in Section 2.4. The relative position of commands and functions within the tables defines the token value. Therefore, if additional tables are implemented then the corresponding tokens will follow in numerical order from the last command in the last list.

7.2.5 The entries in the command table list the valid command names, e.g., ENDFORNEXTDATA... There is no delimiter between each entry. The last character of each individual command has the top bit set - refer to Section 7.7 for a sample new command.

7.2.6 The list of addresses that corresponds with each table points to the address of the code that will handle the specific command. This code will perform the syntax checking, unless catered for in the "handler" routine, and execute the command or function as required.

7.3 IMPLEMENTING NEW TABLES

0134	Number of entries in new command table
0135-0136	Pointer to start of new command table
0137-0138	Pointer to start of new command table handler routine
0139	Number of entries in new function table
013A-013B	Pointer to start of new function table
013C-013D	Pointer to start of new function table handler routine

7.3.1 The above locations define the pointers that must be set up to handle new command and function tables. It is not necessary to implement both tables together. For example, one new command only may be implemented by setting up a new command table with one entry but leaving locations $0139-013D unaltered - the sample new command in Section 7.7 is an example of this.

7.3.2 These locations are exactly the same in concept to the way the Disk BASIC command and function tables are implemented - locations $012A-012E and $012F-0133, respectively, define the Disk BASIC pointers. These Disk BASIC pointers and handler routines may be used as an example of implementing new command and function tables.

7.3.3 Once the above locations have been set up any command or function that is not handled by the first two sets of tables will be passed to this new handler routine. The new handler routine will check if the command is part of the new table, using a range check on the token value, and handle accordingly or generate a "Syntax Error" message.

7.4 IMPLEMENTING A NEW COMMAND HANDLER ROUTINE

7.4.1 The following routine is similar to the command handler for SSSD Disk Basic:-

CMPA	HITOKN	Highest token value in new list
LBHI	$A80F	"Syntax Error" if not in new table
LDX	#ATABLE	Address of start of table of addresses
SUBA	LOTOKN	Lowest token value in new list - subtracted to match ROM code
JMP	$D40D	Jump to ROM code

7.4.2 The above routine should handle all the entries in the new list but is not set up to link to a further command table. The SSSD Disk Basic routine is structured for an additional table (the above table) and this routine could be similarly structured if desired.

7.4.3 There are some limitations in the commands that may be implemented through this new command table. The new commands may not start with a sequence of letters that is already defined as a command, e.g., it is not possible to implement the command "COMPILE" as "COM" is already defined as a command - it is possible to implement, say, "BCOMPILE" as the sequence "BCOM" is not defined. The existing commands, excluding Disk Basic, that may not be used are defined in Appendix C.

7.5 IMPLEMENTING A NEW FUNCTION HANDLER ROUTINE

7.5.1 Implementing a new function handler routine is similar to implementing a command handler routine. The editing of the new functions may be more complex as the possible parameter values and types must be handled.

7.5.2 The following routine is similar to the Disk Basic function handler. The token values used in this routine are not the same as listed in Appendix C. The ROM code removes the top bit and multiplies the value by two - this is then used as an offset into the address list. The token values used in the following routine have been modified as outlined:-

CMPA	HITOKN	Highest token value with top bit cleared and multiplied by two
LBHI	$A80F	"Syntax Error" if not in new table
LDX	#ATABLE	Address of start of table of corresponding addresses
JMP	$D3B4	Jump to ROM code

7.5.3 Similar limitations apply to the possible functions that may be implemented as outlined in Section 7.4.3.

7.6 EDITING NEW COMMANDS AND FUNCTIONS

7.6.1 The code to edit new commands and functions is extremely variable and it is not possible to describe all possibilities. The best method of discovering the editing routines is to select an existing command that has similar editing requirements to the new command and inspect the relevant area of the ROM (refer Appendix C for ROM addresses for each command).

7.6.2 The following routines are included as they may assist - the possible parameter values should be tested to ensure the routine will work for all possible combinations of values - all the routines work on the current line of the BASIC program or the statement in the command line area (locations $0278-0377):-

JSR $B5	- Returns next character from statement in A register - a BEQ test will jump if no more characters are available
JSR $BB	- Returns current character in A register
JSR $C4	- Checks if next character is a comma - gets next character after the comma in A register or generates "Syntax Error" message
JSR $CA	- Similar to JSR $C4 except that character checked for is in the B register
JSR $C7	- Accumulator variable type check - following tests are used after JSR - BVS=single BHS=double BLO=single or integer BEQ=string BMI=integer
JSR $AF81	- Place numeric value from command line into B register
JSR $AFBF	- Place numeric value from command line into X register
JSR $A7FB	- Similar to JSR $C4 except that character checked for is ")"
JSR $A7FE	- Similar to JSR $C4 except that character checked for is "("
JSR $A7F6	- Place numeric value enclosed in quotes into accumulator
JSR $AF84	- Converts value in accumulator to B register - must be positive and in the range 0-255
JSR $A831	- Place numeric value from command line into accumulator

7.7 SAMPLE NEW COMMAND

7.7.1 The following code implements a new command to set the default device number. The syntax is the same as for the FILES command, e.g., DEVICE "1:" will set the default device to drive 1. The parameter may also be a string variable, e.g., DEVNO$="1:":DEVICE DEVNO$. The code should be located in a "safe" area. The code sets up the pointers to the new table and returns control to

BASIC. Once this has been executed the new command will be vaild.

```
        LDA    #$1       Number of entries in new table
        STA    $0134
        LDX    #CTABLE   Pointer to command table
        STX    $0135
        LDX    #CHNDLR   Pointer to command handler routine
        STX    $0137
        JMP    $A438     Jump back to BASIC
CHNDLR  CMPA   #$E6      Highest token value in new table -
                         SSSD  Disk  Basic  tokens go up to
                         $E5
        LBHI   $A80F     "Syntax Error" message
        LDX    #ATABLE   Pointer   to   list  of  addresses
                         matching command table
        SUBA   #$E6      Subtract  lowest  token  value  to
                         match ROM code
        JMP    $D40D     Jump to ROM code
CTABLE  FCC    /DEVIC/   Command  table - last character of
                         each command has its top bit set
        FCB    $C5       "E" with top bit set
ATABLE  FDB    DEVCDE    Address of DEVICE code
DEVCDE  JSR    $E402     Label processing routine - handles
                         device names
        LDA    $0206     $E402  routine  sets device number
                         in location $0206
        STA    $011B     Store new default device number in
                         location $011B
        RTS              End of DEVCDE
```


8.1 INTRODUCTION

8.1.1 This section details a few apparent bugs, some anomalies or "funnies" that may cause problems and some "fixes". The "fixes" relate mainly to avoiding the problem except for the SSSD Disk Basic EOF test which is patched in the DOSMOD program detailed in Appendix F. The "fixes" for some of the error conditions invlove modifying the system work area when the error condition occurs.

8.1.2 Most of the following suggestions for overcoming problems are only intended to allow the data or program in memory to be recovered and the system should be re-initialised as soon as the critical data or program has been saved.

8.1.3 All the disk related problems and fixes have been encountered with SSSD Disk Basic and may not relate to other Disk Basic versions.

8.2 PEN 9/CASSETTE BUFFER CONFLICT

8.2.1 The BASIC manual (Section 2-10-2) states that PEN areas may range from 1 through 9. The PEN statement will allow areas between 1 and 9 to be defined. Use of PEN 9 conflicts with the cassette buffer and using both concurrently will cause problems. The conflict appears to be that the PEN work area is structured for eight defined areas only but the BASIC command will allow nine areas to be defined - the PEN 9 details overwrite the first few bytes of the cassette buffer, and vice-versa.

8.2.2 The ON PEN statement will allow nine line numbers in total with the first being for the undefined area, i.e., the ON PEN statement only caters for eight defined areas. Use of the ON PEN statement with the allowable nine parameters will overwrite some of the PEN 9 details, and vice-versa.

8.2.3 Due to the above conflicts it is suggested that PEN 9 should not be used - PEN 1 to 8 are handled correctly and will not conflict with the cassette buffer or the ON PEN statement.

8.3 "DEVICE IN USE" ERROR

8.3.1 Under certain unexplained conditions the above message will be generated when the cassette or COMO have been used but all files have been closed. The first suggested solution is to enter "CLOSE" again. If the error still occurs then it is likely that the open files list has been corrupted and BASIC thinks the file has previously been closed. For cassette and COMO the solution is to set location $01F0 to zero (refer Appendix A for valid values).

8.4 "FILE ALREADY OPEN" OR "FILE NOT OPEN" ERROR

8.4.1 If the above error message occurs when the file is not open or closed, respectively, then it is possible that the open files list has been corrupted. Section 3.7 details the way the open files list is structured and it may be possible to set the relevant byte to the appropriate value.

8.4.2 For disk files it is possible to get the "File Already Open" message when trying to save to disk. This may be caused by ABORTing the SAVE and may be fixed by executing a CLOSE. If other disk accessing is performed before the CLOSE then the file will remain open and the problem outlined in Section 8.12 is likely to occur. If the CLOSE does not work then the file should be saved under a different name or to a different drive and the system re-initialised. It may be possible to set the byte in the open files list and execute a CLOSE but the correct value may be difficult to ascertain - it is safer to recover as outlined above and re-boot.

8.5 "OUT OF MEMORY" ERROR

8.5.1 This error seems to occur occasionally when there is obviously sufficient memory available. The ROM checks that there is approximately #$3A bytes between the top of free space (location $23-24) and the bottom of the stack (copy in location $1B-1C). These locations should be checked to ensure they have not been corrupted. Location $23-24 may be reset by executing a "CLEAR" command and ensuring location $1F-20, $21-22 and $23-24 are set to the same value.

8.5.2 This error may also occur when executing the "CLEAR" command. In this case it is possible that location $03D8-03D9 has been corrupted. This location should point to the top of available RAM - just below the physical top of RAM or the start of Disk Basic. If it has been corrupted then reset it to some value near the top of available RAM, save the program and re-boot the system.

8.6 RANDOM GRAPHIC CHARACTERS ON SCREEN DISPLAY

8.6.1 Under certain conditions characters on the screen display will change mode and appear as block graphic characters. This is especially noticeable with programs that have a large string space area - CLEAR 12000 or greater. Section 5.2 details some possible reasons for this and suggests a possible way to minimise the problem from within a machine code program. It does not seem possible to fix the problem while running a BASIC program.

8.7 CASSETTE LOADING PROBLEMS

8.7.1 Some cassette recorders appear to have problems reading tapes and generate frequent "Device I/O" errors. With some recorders it has been noted that the record lead causes some form of interference when loading (refer Section 3.2.4.3). If a recorder is giving trouble with reading then try removing the record lead when loading.

8.8 "EOF" TEST FOR SSSD DISK BASIC

8.8.1 There is a bug in SSSD Disk Basic in that the value returned when "EOF" is true is not correct. This will only cause a problem with a "IF NOT EOF(x)..." statement. The value returned is "00FF" instead of "FFFF" and will not give zero when complemented for the "NOT" condition.

8.8.2 The DOSMOD program detailed in Appendix F includes a patch for SSSD Disk Basic to fix this problem - it changes the return jump into the ROM to ensure a "SEX" (sign extend) instruction is executed ($E8CD in lieu of $ACE6).

8.9 AUTO-BOOT FOR SSSD DISK BASIC

8.9.1 The 32K version of SSSD Disk Basic includes a facility for automatically running the file with a blank file name on initialisation. This feature is not included with 40K SSSD Disk Basic. It appears as if this auto-boot feature has been added as a patch as it is implememted by executing two BASIC commands from machine code - 'Color 0,0' and 'R."'. One problem with this method is that the system will be left in 'COLOR 0,0' if there is no file with a blank file name on the disk.

8.9.2 The DOSMOD program detailed in Appendix F fixes this problem and also implements the auto-boot for 40K SSSD Disk Basic. The method used is to jump into the ROM at the point where the 'R."' command is executed ($E6C5).

8.10 KEYBOARD INPUT WHILE SAVING TO DISK

8.10.1 Keyboard input while saving to Disk will generally cause the system to crash or "hang". The only solution in some cases is to switch off and on again. The reason for this has not been investigated in any detail. The obvious solution is not to use keyboard input while saving to Disk.

8.11 "DISK FULL" ERROR - SSSD DISK BASIC

8.11.1 This message will appear when the disk is full but may continue to be generated after sufficient space has been created. The solution is to CLOSE and KILL the file that caused the message before attempting to create space or do other processing.

8.12 "FILES" CHANGING SIZE - SSSD DISK BASIC

8.12.1 Under certain conditions the number of groups shown on the "FILES" display changes when the file has not been accessed. The directory appears to be corrupted as the files will generally not load. In addition it is possible that the disk may be permanently corrupted and files may be lost - this does not occur in all cases.

8.12.2 This problem is caused by the way Disk Basic handles the file control blocks. The system uses the file control block in memory if a file is open on that drive. If a different disk is inserted the file details displayed will relate to the file control block in memory and will not relate to the disk in the drive.

8.12.3 Therefore, a new disk should not be inserted in a drive if there is an open file. The FILES command will give no indication that it is using the data in memory. It is possible to patch Disk Basic to return a "File Already Open" message if there is a file open on the relevant drive. This has one problem in that the FILES command may not be used for a drive that has any files opened. This restriction may be preferable as it will minimise the problem with file sizes being corrupted and will give positive indication that a file is open.

8.12.4 The patch for Disk Basic is simple in that the BNE $7735 at location $7776 should be changed to a BNE $775E (generates a "File Already Open" message). To achieve this it is necessary to change location $7777 from $BD to $E6, i.e., adjust the PC relative jump value. This patch should work for both 32K and 40K Disk Basic. The DOSMOD program detailed in Appendix F includes a prompt to allow or ignore this patch.

9.1 INTRODUCTION

9.1.1 This section details several useful routines within the ROM together with some notes on various points of general interest. The majority of the useful ROM routines have already been detailed in other areas of the notes. The routines detailed in this section are ones that do not relate specifically to othe areas of the notes.

9.2 EDITING ROUTINES

9.2.1 The following routines may be used for data editing or as part of the editing routines for additional commands and functions.

- JSR $AD00 - returns the length of a string in the B register. The string may be terminated by a specified delimiter or by a zero byte. Set the starting and ending delimiters in the A and B registers and set the X register to point to the first byte of the string after the delimiter.
- JMP $A3A7 - this routine may be used to generate an error message and return to the BASIC command level. The error message number should be placed in the B register.
- JSR $E86E - this routine may be used to check if the file number in location $9E is a disk file. A BMI after the routine will branch if it is a disk file.
- JSR $E8D4 - this routine edits the file number in the B register for the range 0 to 16. If it is outside this range then an error message is generated and control is returned to the BASIC command level.

JSR $E636 - this routine edits ASCII characters for a valid file number parameter in the form #FF or FF where FF is in the range 1 to 16. If the parameter is invalid then an error message is generated and control is returned to the BASIC command level. Location $BC should be set to point to the first character of the parameter.

JSR $E87B - this routine checks if the file number in location $9E is open for output - if not, an error message will be generated and control returned to the BASIC command level.

JSR $E87E - this is identical to the above routine except that the file is checked for open to input.

9.3 DATA CONVERSION ROUTINES

9.3.1 The following routines relate to data conversion or manipulation of numerical values.

JSR $C45E - this routine sign extends the signed byte in the B register. The two byte result is returned in BASIC's accumulator - locations $58-59.

JSR $C5A6 - this routine converts a string of ASCII characters to a numeric value in BASIC's accumulator. Location $BC should be set to point to the byte before the start of the ASCII character string. Care should be taken in using this routine as the variable type of the returned value in BASIC's accumulator will depend on the size of the numeric value. Refer to Section 2.6 for details on BASIC's accumulator.

9.4 BLOCK MOVE ROUTINE

9.4.1 There is a general block move routine within the ROM that can move blocks of memory up to 255 bytes in length.

JSR $FC48 - set the X register to point to the start of the source area and the U register to the start of the destination area. The number of bytes to be moved should be set in the B register. There is an absolutely identical routine at location $EC41.

9.5 MEMORY INITIALISATION ROUTINES

9.5.1 The following routines may be used to set blocks of memory to constant values.

JSR $F15A - set the X register to point to the start of the area to be initialised. Set the A register to the constant value to be placed in the block of memory and the B register to the number of bytes to be initialised.

JSR $FC8A - this routine may be used to set blocks of memory to repeating patterns of three bytes. It may also be used to set up to 3*255 bytes to a constant value (set all three bytes to the same value). This routine is used by the ROM to initialise areas of memory to default jumps, e.g., JMP $AC06. The first byte of the three should be set in the A register and the remaining two in the X register. The number of groups of three bytes should be set in the B register.

9.6 "OTHER" ROUTINES

9.6.1 WAIT LOOP

JSR $F7BF - this routine may be used as a wait loop. Set the A register to the wait value - the A register is decremented by a DECA until it is zero.

JSR $E953 - this routine is also a wait loop. Set the Y register to the wait value - the Y register is decremented by a LEAX -1,Y until it is zero.

9.7 RS232 SIGNALS AND CABLES

9.7.1 The RS232C control signals used by COMO are RTS, CTS and CAR. The internal switches in CS4 can be set to ignore these signals and this is probably the best option if it can be used (refer Section 9.7.2).

9.7.2 When COMO is opened for output RTS is raised. Data will only be output if CTS is not low (OFF), i.e. either high (ON) or open circuit. If a printer uses a control line to control data flow, the line should be connected to CTS on the Peach.

9.7.3 Input data will only be read if CAR is high (ON). If CAR is high (ON) or open circuit then SPACE (continuous zero) condition on input for more than one character period will cause an I/O error. This can be seen by entering TERM with nothing plugged into the RS232C port. If COMO is open for input and CAR drops then output will hang after the next byte. Output will resume after CAR comes up and a character is input.

9.7.4 A half duplex modem line cannot be used with the standard COMO drivers since RTS is held permanently high and output is inhibited if CAR drops.

9.7.5 Diagram 1 is a loopback plug which can be used for I/O testing on COMO. The loopback plug shown is for general use, including synchronous operation. The 17-24 connection is not required for asynchronous operation as used by COMO. DSR and DTR are not used by COMO so the 6-20 connection is not required. If CTS and CAR are internally switched to ground only 2-3 need be connected.

9.7.6 The RS232C port is designed for use with a modem. For connection to another terminal or printer a crossover cable is required. A general purpose crossover cable is described in Diagram 2. This cable will not work with a device which requires DSR since DTR is not raised by COMO. A less universal cable is described in Diagram 3. Again the 17-24 connections are only required for synchronous operation. The 6-20 connection is not required on the COMO side and the 4-5-8 connections are not required if CTS and CAR are internally grounded. The cable in Diagram 2 will not work for a printer which uses a control line to control data flow (refer Section 9.7.2). The cable should have a male plug on the COMO end. The sex of the plug on the other end is determined by the device at the other end.

Loopback Plug

```
Peach

SIGNAL PIN

TD     2 ---+
            |
RD     3 ---+

RTS    4 ---+
            |
CTS    5 ---+
            |
DSR    6 -----+
            | |
GND    7    | |
            | |
CAR    8 ---+ |
              |
TC    15      |
              |
RC    17 ---+ |
            | |
DTR   20 -----
            |
TC    24 ---+
```

Diagram 1

Universal Crossover Cable

```
 Term 1                          Term 2

SIG PIN                          PIN SIG

TD  2 ---------\/--------- 2 TD
               /\
RD  3 --------/  \-------- 3 RD

RTS 4 -+--\          /--+- 4 RTS
       |   \        /   |
CTS 5 -+    \      /    +- 5 CTS
             \    /
DSR 6 ----\   \/   /---- 6 DSR
           \  /\  /
GND 7 ------------------ 7 GND
          /\   /\
CAR 8 ---/  \/  \----- 8 CAR
             /\
TC 15       /  \        15 TC
           /    \
RC 17 -------\  /-------17 RC
          /  \/  \
DTR 20---/   /\   \---20 DTR
            /  \
TC 24 ------/    \------24 TC
```

Diagram 2

Peach Crossover Cable

```
Peach                       Term 2  RS232C Pin Usage

SIG  PIN                   PIN  SIG   TERM MODEM NAME

TD   2 ---------\/--------- 2  TD    out  in    Transmit
                /\                              Data
RD   3 --------/  \-------- 3  RD    in   out   Recieve
                                                Data
RTS  4 --+              +-- 4  RTS   out  in    Request
         |              |                       To Send
CTS  5 --+              +-- 5  CTS   in   out   Clear
         |              |                       To Send
DSR  6 -----+        +----- 6  DSR   in   out   Data Set
         |  |        |  |                       Ready
GND  7 -------------------- 7  GND              Signal
         |  |        |  |                       Ground
CAR  8 --+  |        |  +-- 8  CAR   in   out   Carrier
            |        |                          Detect
TC  15      |        |     15  TC    in   out   Transmit
            |        |                          Clock
RC  17 ------\      /------17  RC    in   out   Receive
            |  \  /  |                          Clock
DTR 20 -----+   \/   +-----20  DTR   out  in    Data Term
                /\                              Ready
TC  24 -------/  \-------24  TC    out  in    Transmit
                                                Clock
```

Diagram 3

9.8 SAMPLE ACIA PROGRAM - TELETYPE "BREAK" FUNCTION

9.8.1 The routine listed on the following pages implements the Teletype break function for COMO output. The break is generated by the BREAK key, using an interrupt routine, whenever COMO is open for output (as it will be in TERM).

```
     6F00                 ORG    $6F00
                   * A ROUTINE TO IMPLEMENT THE BREAK FUNCTION
                   * FOR THE TERM COMMAND
                   *
                   * WRITTEN FOR THE HITACH MB6890 (PEACH)
                   * H. VICCARS - 6809 ASSEMBLER - OCT 1982
                   * POSITION INDEPENDENT CODE
                   *
                   * THIS PROGRAM SETS UP THE NMI INTERRUPT
                   * AND USES THE BREAK KEY TO GENERATE THE
                   * ASYNCHRONOUS BREAK (INTERRUPT) FROM THE
                   * ACIA IN AN INTERRUPT ROUTINE
                   * FOR COMO WHEN OPEN FOR OUTPUT
                   *
                   * THE FIRST PART SETS UP THE INTERRUPT OPERATION
                   * THE SECOND PART IS THE INTERRUPT ROUTINE
                   *
                   * SYMBOLIC EQUATES
          FFC8     KBNMI  EQU   $FFC8         KEYBOARD NMI REGISTER
          01E0     COMWRK EQU   $1E0          POINTER TO COMO WORK AREA
          0109     NMIVEC EQU   $109          NMI VECTOR LOCATION
          01CA     KBWRK  EQU   $1CA          POINTER TO KBD WORK AREA
          FFE0     KBHWR  EQU   $FFE0         KEYBAORD REGISTER
                   *
```



```
                   * SET UP INTERRUPT VECTOR AND ENABLE BREAK NMI
                   *
6F00 B60109        INIT   LDA   NMIVEC        GET ORIGINAL NMI INTSR
6F03 BE010A               LDX   NMIVEC+1      GET NMI ADDRESS
6F06 A78D004B             STA   EXIT,PCR      NMI ROUTINE EXIT
6F0A AF8D0048             STX   EXIT+1,PCR    NMI EXIT ADDRESS
6F0E 867E                 LDA   #$7E          JUMP INSTRUCTION
6F10 308D0013             LEAX  NMIRTN,PCR    GET ROUTINE ADDRESS
6F14 B70109               STA   NMIVEC        SET NMI JUMP
6F17 BF010A               STX   NMIVEC+1      SET NMI ADDRESS
6F1A BE01CA               LDX   KBWRK         POINT AT KBD WORK AREA
6F1D A603                 LDA   3,X           GET CURRENT KB SETTING
6F1F 8A80                 ORA   #$80          ALLOW BREAK KBNMI
6F21 A703                 STA   3,X           NEW VALUE FOR KB SETTING
6F23 B7FFE0               STA   KBHWR         SET HARDWARE REG
6F26 39                   RTS
                   *
```

```
* NMI VECTOR ROUTINE - CHECK COMO IS OPEN FOR
* OUTPUT. EXAMINE BREAK KEY AND SEND BREAK -
* IF SET EXIT TO NORMAL NMI VECTOR
6F27 BE01E0   NMIRTN LDX  COMWRK    POINT TO COMO WORK AREA
6F2A A68815          LDA  $15,X     COMO STATUS
6F2D 8520            BITA #$20      COMO OPEN FOR OUTPUT?
6F2F 2724            BEQ  EXIT      IF NOT EXIT
6F31 B6FFC8          LDA  KBNMI     BREAK INPUT?
6F34 2A1F            BPL  EXIT      IF NOT EXIT
6F36 A6881A   BRKAGN LDA  $1A,X     GET COMO OPTIONS
6F39 8A60            ORA  #$60      SET BREAK FUNCTION
6F3B A79817          STA  [$17,X]   WRITE TO ACIA
6F3E 8D18            BSR  WAIT
6F40 8603            LDA  #$03      MASTER RESET VALUE
6F42 A79817          STA  [$17,X]   RESET ACIA
6F45 A6881A          LDA  $1A,X     GET COMO OPTIONS
6F48 8A80            ORA  #$80      ALLOW INTERRUPTS
6F4A A79817          STA  [$17,X]   RESTORE ACIA
6F4D B6FFC8          LDA  KBNMI     CLEAR BREAK REGISTER
6F50 B6FFC8          LDA  KBNMI     IS KEY STILL PRESSED?
6F53 2BE1            BMI  BRKAGN    IF SO REPEAT BREAK
6F55 0003     EXIT   RMB  3         ORIGINAL NMI VECTOR HERE
*
* WAIT LOOP - DETERMINES PERIOD OF BREAK - > 200 MS
6F58 108E7FFF WAIT   LDY  #$7FFF    SET COUNTER
6F5C 313F     LOOP   LEAY -1,Y      COUNT DOWN
6F5E 26FC            BNE  LOOP
6F60 39              RTS
```


9.9 EDITING MODE SETTINGS

9.9.1 Many programs, both BASIC and machine code, require specific mode settings to be set. This normally requires a NEW ON to be executed before the program is run. A better method is to include code to specifically test the swith settings at the start of the program and automatically execute a NEW ON if it is not the correct mode. If the auto-boot feature is used then the program will automatically handle setting the corrrect mode.

9.9.2 Some programs use this method but unconditionally re-boot to a set mode, e.g., NEW ON 7. Whilst this works for the particular program it may unnecessarily change some settings and may cause confusion if the system is not re-booted after finishing with the particular program.

9.9.3 The preferable solution is to only reset the parameters that are specifically required. This may be done by masking the switch settings to set specific switches to a "1" or "0". Location $0248 contains a copy of the switch settings. Section 2-12-2 of the BASIC manual details the switch settings and NEW ON values.

9.9.4 The following BASIC code checks the switch settings for switch 1, 3, 4 and 5. It checks that the PF keys are not displayed (switch 5=0), low resolution mode is set (switch 4=1), width is 80 (switch 3=1) and BASIC mode is set (switch 1=1). The &H1D mask selects switches 1, 3, 4 and 5. The &H0D test checks for the desired pattern of "0" and "1" bits. The switch settings are changed by the AND &HE2 mask (selects the bits that do not require changing) and the OR &H0D mask (unconditionally sets switch 5 to "0" and switches 1, 3 and 4 to "1").

```
10   ISW = PEEK(&H0248)
20   IF (ISW AND &H1D) <> &H0D THEN
        NEW ON (ISW AND &HE2 OR &H0D)
30   ' Start of program code - correct mode is set
```

A.1 INTRODUCTION

A.1.1 The following notes relate to the locations below the start of BASIC or user program area. Notes on the following format are included at the end of the appendix (Section A.4).

A.2 BASIC AND SYSTEM WORK AREA

HEX ADDRESS		DESCRIPTION
0006		Accumulator variable type - 2=int 3=char 4=single 8=double
0010-0011		Return address for certain BASIC functions
001B-001C		Copy of Stack Pointer
001D-001E		Start of BASIC
001F-0020		Start of Variables
0021-0022		Start of Arrays
0023-0024		Start of free space
0025-0026		Top of Stack - lower limit of string space
0027-0028		Next available slot in string space
0029-002A		Last used string
002B-002C		Top of String space
002D-002E		Current line number
002F-0030		Line number in error or last line executed
0031-0032		General storage
0033-0034		Pointer to current part of BASIC program - used to restart following CONT command
0039-003A		Pointer for next DATA statement
0056-005D		BASIC's accumulator (3-1-3 of BASIC manual)
005E-005F	??	Negative indicator
0073	??	Something to do with IF and ELSE
007C		AUTO command flag
007D-007E		AUTO starting line number
007F-0080		AUTO increment value
0081-0082		Starting line no.) LIST/DELETE
0083-0084		Ending line no.) Command
0088		BASIC trace flag 0=TROFF $4F=TRON

008B		BASIC error message no. (ERR value)
008C-008D		Line no. in error (ERL value) $FFFF=Direct Statement
008E-008F		Used to store stack pointer while running BASIC
0090		ON ERROR flag - 0=no error $FF=error
0091-0092		Pointer to line to RESUME
0093-0094		ON ERROR pointer
009E		File Number (1-16 for BASIC - 0-17 internally with 0 and 17[$11] reserved for system use)
009F		EOF flag for $E804 input routine - $FF=EOF
00A1		LINE option 00=PRESET 01=PSET $C8=Char. option
00A2		WIDTH - $28=40 col. $50=80 col.
00A3-00A4		Start of current screen page
00A5-00A6		End of current screen page
00A7		0=40 col. low res. 1=80 col. low res. 2=40 col. high res. 3=80 col. high res.
00A8-00A9		Starting horizontal position - (Various screen commands)
00AA-00AB		Starting vertical position
00AC-00AD		Ending horizontal position
00AE-00AF		Ending vertical position
00B5-00CC		Code for next part of BASIC line - following JSR's are used to edit BASIC command (Refer Section 7.6)
		B5 - Next character in A register
		BB - Current character in A register
	??	C1 - Adds B to X (X=X+B)
		C4 - Checks next char. for ',' and get next character - Syntax Error if not ','
		C7 - Accumulator variable type check - following tests are used after JSR - BVS=single BHS=double BLO=single or integer BEQ=string BMI=integer
		CA - Check next char. with B register - same result as C4

00FD	ABORT flag - 55=Warm restart XX=Power on restart
00FE-00FF	Warm restart address ($EF0E)
0100	SWI3 interrupt jump
0103	SWI2 interrupt jump
0106	SWI interrupt jump
0109	NMI interrupt jump
010C	IRQ interrupt jump
010F	FIRQ interrupt jump
0112-0114	LPT0-LPT2 CR/LF options (1-4-2 in BASIC manual)
0115	Cursor flag - 00=off $20=static $40=on fast $60=on normal
0116	BASIC's copy of colour register ($FFD8)
0119	No. of $FF sync bytes to be output in front of CASO block if motor already on [EE7E]
011A	No. of $FF sync bytes to be output in front of CASO block if motor turned off [EE85]
011B	Default device number (02=Cassette $80=Drive 0)
011D-011F	Jump for ON ERROR/COM/KEY [E1BB]
0120	Number of entries in BASIC command table
0121-0122	Pointer to BASIC command table
0123-0124	Pointer to addresses matching command table
0125	Number of entries in BASIC function table
0126-0127	Pointer to BASIC function table (tokens = 'FF XX')
0128-0129	Pointer to addresses matching above table
012A	Number of Disk Basic Commands
012B-012C	Pointer to Disk Basic command table
012D-012E	Pointer to Disk Basic command handler
012F	Number of Disk Basic functions
0130-0131	Pointer to Disk Basic function table
0132-0133	Pointer to Disk Basic function handler
0134-013D	Locations for additional command or function tables (refer Section 7)

0148		Go address from LOADM or EXEC
014A-015D		USRx Jump Addresses - x=0 to 9
015E-0160		LOAD/LOADM jump if not CASO or Disk [E71E]
0161-0163		DOS jump for FILES
0164-0166		DOS jump for OPEN
0167-0169		DOS jump for CLOSE
016A-016C		DOS jump for character output(JSR $E820)
016D-016F		DOS jump for character input (JSR $E804)
0170-0172		DOS jump for $E84B ?? routine [E857]
0173-0175		DOS jump for EOF, LOF routines [E8A9]
0176-0178	??	DOS jump for INPUT # [D0FD]
0179-017B	??	"Device Unavailable" jump [????]
017C-017E		WHILE jump - gives "Syntax Error"
017F-0181		WEND jump - gives "Syntax Error"
0182-0184		Jump for command line input [F4DF]
0185-0187		Jump for end of BASIC statement [D3BA]
0188-018A		Jump for Disk Basic functions [D449]
018B-018D		Jump for MON command [DCE6]
018E-0190		Jump for error messages [A3A9]
0191-0193		Jump for Disk Basic to trap error messages [A3AC]
0194-0196	??	Jump for ?? [A787]
0197-0199	??	Jump for ?? [D26A]
019A-019C		Jump for BASIC function address table [D39F]
019D-019F	??	Jump for ?? [D1D6]
01A0-01A2		Jump for function handler code [D360]
01A3-01A5		Jump for initialising variables [A516]
01A6-01A8		Jump for TERM command [EC49]
01A9-01AB	??	Jump for ?? [A73A] (Jumps to DOS)
01AC-01AE	??	Jump for string handling [DE50]
01AF-01B1		Jump for CTRL C/CTRL D [ED94]
01B2-01B4		Jump for CTRL S key [F814]
01B5-01B7		Jump for variable type checking routine [AAA2]
01B8-01BA	??	Jump for string handling [AD98]
01BB-01BD		Jump for $E402 file label routine [E405]
01BE-01C7		TAB map (refer Section 5.9)
01C8-01C9		Pointer to end of screen RAM
01CA-01CB		Pointer to keyboard work area and buffer

01CC-01CD		Pointer to start of PF keys [F63A, F84C]
01CE-01CF		Pointer to L/Pen screen co-ordinates [E18F, E1C6]
01D0-01D1		Pointer to Cassette buffer [E23E, E290, E2AA]
01D2-01D3		Pointer to device list [E491, E60C, E64A]
01D4-01D5		Pointer to open files list [E5F6, E658, E6BD]
01D6-01D7		Pointer to Light Pen status flags
01D8-01D9		Pointer to COMO-COM4 status flags [EBC1]
01DA-01DB		Pointer to PF Key status flags [FAAB]
01DE-01DF		Pointer to screen line map [F154, F1A7, F4D9]
01E0-01E9		Pointer to COMO-COM4 jump table
01EC		CASO buffer pointer
01EE		CASO block type $FF=trailer 0=header 1=data
01EF		CASO block length
01F0		$FF=COMO 01=CASO input 02=CASO output
01F1		Number of bytes in CASO buffer
01F2-01F3		Next slot in CASO buffer
01F4		Used for open - set to same as $01FC (File format) and placed in CASO header block [E24E, E26D, E66E...]
01F5		Used for CASO I/O error [EE24, EE2A]
01F6		Set equal to $01F4 after open. Controls motor off at end of data block. [E270, EEBF]
01F7		CASO motor - 0=off $FF=on $80=on(load/save)
01F8		CASO BASIC mode flag - 0=Immediate command (print messages) $FF=Program mode (suppress messages) [E343, E37E, E391
01F9	??	$FF=Cassette buffer in use [B09D..., D10B, E336]
01FA		I/O Mode - $10=input $20=output
01FB		File Type 0=BASIC 1=Data 2=Machine code
01FC		File format 0=Binary $FF=ASCII

01FD		Label length (number of characters in label)
01FE-0205		Label - blank fill to 0205
0206		Device no. 0=KYBD: 1=SCRN: 2=CASO: 3=COMO: 4=LPTO: $80=0: $81=1: $82=2: $83=3:
0207-020A		COMO options , e.g., "S8N2"
020E		02=RUN 03='R' option of LOAD/LOADM [E6E8, E714]
020F		MERGE indicator $FF=MERGE [E6EB]
0210		LOADM indicator 00=LOADM XX=LOAD/MERGE/RUN [E6C7, E6CF, E6DC, E6F1, E729]
0211-0212		LOADM offset value [E6E5, E703, E7D8]
0213	??	Reset BASIC flag [A3D2, A561, E75D, EDAE]
0214		BASIC's time flag - 0=time on 1=time off (BASIC only) [E158]
0215		60 Hz timer counter [DAB2,D97B]
0216-021B		BASIC's time - one byte per digit
021C-0221		BASIC's date - one byte per digit
0222-022C		PEN function work area [E12D, E144]
0238		Screen column no. (0 - 79)
0239		Screen line no. (0 -24)
023A		Bottom line of screen (for cursor back to top)
023B		Console top line number
023C		Console bottom line number
023D		PF Key flag - 00=Key off 01=Key on
023E		Shift key pressed when PF Keys displayed
023F	??	Something to do with CRTC register $38=80 col. $0E=40 col. [E045, FOC6]
0240		MON option flag
0241	??	Screen line number - MON command ?
0242	??	Screen col. number - MON command ?
0243		Last character input to line buffer (Not used while running BASIC program)
0244		BREAK/CTRL C/CTRL D flag - $FF=BREAK 03=CTRL C 04=CTRL D
0245		CTRL D flag - 04=CTRL D

0246		CTRL S flag - $13=CTRL S
0247		Copy of Interlace Select register ($FFD6) [DEFF]
0248		Switch settings (NEW ON value)
0249		Copy of Mode Select register ($FFD0)
024A-024B		INPUT WAIT time - multiples of 60Hz [A3B7, D04A..]
024D-024E		INPUT WAIT line number address [D04C, D07C]
024F-0250		MON - last value for D or M option
0251		Stack Pointer - temporary storage for MON command
0253-0259	??	Random number area [CC6D...]
025A-0273		Variable type map - one byte for each letter of the alphabet - value=variable type - used for DEF statement
0274-0275	??	Points to start of screen RAM
0278-0377		BASIC's 255 byte command line
037A-03A2		Conversion work area - ASCII-decimal conversions
03A3-03A2		Last variable name referenced [AA5E, AABC]
03A3-03B2		Last variable name referenced
03B3-03B5		Pointer to last string referenced
03D3		'Break' flag - +ve='Ready' -ve='Break Ready' [A58F, A5A3]
03D4	??	'?' prompt flag for INPUT statement [CFDC, D00C, D062]
03D5		COM/PEN/KEY indicator - incremented by 1 on COM/PEN/KEY input and used to trigger ON COM/PEN/KEY handling (decremented by one when input handled)
03D6		CTRL C/S inhibit flag (not BREAK) [F759]
03D7		TERM duplex flag - 00=Full 02=Half
03D8-03D9		Top of available memory [DE92]
03DA-03DB		Last horizontal point from LINE Command
03DC-03DD		Last vertical point from LINE Command
03DE	??	Something to do with PAINT command
03E4-03E5	??	Something to do with PAINT command

03E6-03E7		Max. horizontal pixels - PAINT Command - value depends on resolution
03E8-03E9		Max. vertical pixels
03EA		LINE/PAINT/PSET color option - 1X=graphic 0X=character X=color
03EB		Character for LINE command character marking option
03EC		Bounbary color for PAINT command
03ED-03EE	??	Pointer to screen handler for PAINT
0400-[01C8]		Screen RAM (end address depends on mode)
	**********	SCREEN RAM AREA ***********************
[01CA]		Keyboard status byte - following bit values (0=Off 1=On) 7 - CAPS Lock 6 - SHIFT 5 - K/H(green) 4 - K/H(red) 3 - CTRL 2 - GRAPH 1 - Not Used 0 - Not Used
+$1		Last value input from $FFE0
+$2		Used for temporary storage
+$3		Copy of value written to KB register ($FFE0)
+$4		Number of keys pressed (max 10)
+$05-$0E		Value from $FFE0 for each key pressed - the top bit is set while the key is pressed and is turned off when the key is raised
+$0F-$18		Auto-repeat counter for each key pressed -last key pressed repeats
+$19		Displacement from [01C8]+$1B for next character
+$1A		Displacement from [01C8]+$1B for first character
+$1B-$3A		Keyboard advance buffer (circular buffer - 32 ch)
[01CC]		PF Keys
[01CE]		PEN screen area (1-9) co-ordinates. NOTE! - PEN 9 conflicts with Cassette buffer and may not work correctly when both in use. Refer Section 6.39 for further details - +$0 - Starting horizontal position

```
                     +$1 - Starting vertical position
                     +$2 - Ending horizontal position
                     +$3 - Ending vertical position
                   +$4-5 - Pointer to line for ON PEN
        +$30-$31   Pointer to line for undefined area
[01D0]             255 byte cassette buffer area
[01D2]             Pointers to device jump tables -
                   relative position of two byte addresses
                   defines device number
[01D4]             Open files list - the relative position
                   defines the file number
                     Non-Disk Files - 1X=input 2X=output
                                      X=device number
                     Disk Files     - 9X=input AX=output
                                      CX=random X=number of
                                      disk buffers
                                      remaining available
[01D6]             Light Pen status flags
                      $0 - 00=PEN OFF  $97=PEN ON
                     +$1 - $8E=PEN STOP
                     +$2 - $FF=Pen strike while 'STOP'
              ??   +$3-4 - set to same value as $01D6 ?
[01D8]             COMO-COM4 status flags - same format as
                             PF Keys
[01DA]             PF Key status flags - 5 bytes for each
                          PF Key
                     +$0 - 00=KEY X OFF  $97=KEY X ON
                     +$1 - $8E=KEY X STOP
                     +$2 - $FF=Key X pressed while 'STOP'
                   +$3-4 - Pointer to line for ON KEY (X)
                           GOSUB ....
[01DE]             Screen line map for lines that span more
                   than one screen line. There are 25
                   entries - one for each screen line.
                   $FF=First line  00=Continuation of
                   original line - refer Section 4.3.3
[01E0]             COMO jump table
[01E0]+$12         COMO work area (offsets are from
                                  [01E0]+$12
                     +$00 - No. of characters output to
```



```
                          COMO since last <CR> (Reset by
                          <CR>)
              ??   +$02 - ?? [EA60, EA64, EAFA]
                   +$03 - Open status - 10=input
                          20=output  30=I/O
                   +$04 - Error flag - top bit set on
                          buffer overflow - bits 4 to 6
                          set to value of ACIA status
                          register
                   +$05 - ACIA address
                   +$07 - Default COMO ACIA control
                          register values (not used by
                          TERM)
                   +$08 - Current value of ACIA control
                          register values (except for top
                          bit). See data sheet for
                          details.
                   +$09 - Graphics indicator - used for 7
                          bit input only (top bit is
                          included with input data for 8
                          bit input) - $00 - input is
                          character (top bit cleared) -
                          $FF - input is graphic (top bit
                          is set)
                   +$0A - Cleared on open for output
                   +$0B - Offset for next byte to be
                          placed in buffer
                   +$0C - Offset to next byte to be read
                          from buffer
[01E0]+$1F-$9E  128 byte COMO buffer area
[01E0]+$9F      LPTO jump table and work area (ends at
                [01E0]+$B6)
     +$B1       No. of characters output in current
                line - used for POS function - set to
                zero on <CR> or <LF>
     +$B4       Points to PIA I/O register
[01E0]+$B7-...  Disk file buffers - number depends on
                the number specified at configuration
                time, generally 5 - maximum 16
[001D] ******** USER PROGRAM AREA ***********
       ******** START OF BASIC PROGRAM ******
```

A.3 INDIRECT ADDRESS VALUES BY MODE

ADDRESS	LOW RESOLUTION 40 COL.	LOW RESOLUTION 80 COL.	HIGH RESOLUTION 40 COL.	HIGH RESOLUTION 80 COL.
[01CA]	0800	0C00	2400	4400
+$1	0801	0C01	2401	4401
+$4	0804	0C04	2404	4404
+$05-$0E	0805	0C05	2405	4405
+$19	0819	0C19	2419	4419
+$1A	081A	0C1A	241A	441A
+$1B-$3A	081B	0C1B	241B	441B
[01CC]	083B	0C3B	243B	443B
[01CE]	08DB	0CDB	24DB	44DB
+$30-$31	090B	0D0B	250B	450B
[01D0]	090D	0D0D	250D	450D
[01D2]	0A0C	0E0C	260C	460C
[01D4]	0A2C	0E2C	262C	462C
[01D6]	0A3E	0E3E	263E	463E
[01D8]	0A43	0E43	2643	4643
[01DA]	0A5C	0E5C	265C	465C
[01DE]	0A98	0E98	2698	4698
[01E0]	0AB1	0EB1	26B1	46B1
[01E0]+$1E	0ACF	0ECF	26CF	46CF
[01E0]+$1F-$9E	0AD0	0ED0	26D0	46D0
[01E0]+$9F	0B50	0F50	2750	4750
[01E0]+$B6	0B67	0F67	2767	4767
[01E0]+$B7	0B68	0F68	2768	4768
****************	USER PROGRAM AREA	**********		
[001D]Cassette	0B68	0F68	2768	4768
[001D]Disk File=5	10B6	14B6	2CB6	4CB6
[001D]Disk File=16	16D0	1AD0	32D0	52D0

A.3 NOTES ON APPENDIX A

A.3.1 The use of square brackets in the address column indicates an indirect address, that is, the location inside the brackets contains a pointer to the location referred to above. For example [01CC] points to the address contained in location $01CC ($0C00 for 80 col.

low res.). This method is necessary as the screen RAM area is variable in length, depending on the screen mode. Refer to the above table for the actual addresses corresponding with each mode.

A.3.2 Addresses contained in square brackets in the description field are the addresses where the location is referenced. These addresses are included where the use of the location is unclear. For some locations the number of places where referenced is too many to be included.

A.3.3 The information contained above has been obtained by disassembling the ROM and by experimentation. In some cases it is exceptionally difficult to determine the exact use for a location. Where the use is unclear the information above includes '??'.

B.1 I/O REGISTER MAP

ADDRESS	REGISTER NAME	FUNCTION
FFC0	PIA	Parallel I/O
FFC4	ACIA	Serial I/O
FFC6	CRTC	CRT Controller
FFC8	KBNMI	KYBD NMI flag
FFC9	DIPSW	Switch Settings
FFCA	TIMER	Timer flag
FFCB	LPFLG	L/Pen flag
FFD0	MODE SEL	Screen mode, etc.
FFD1	TRACE	Trace counter
FFD2	MOTOR	Cass. Motor on/off
FFD3	MUSIC SEL	Music Select
FFD4	TIME MASK	Timer enable
FFD5	L/PEN MASK	L/Pen enable
FFD6	INT'LCE SEL	Interlace enable
FFD7	BANK SEL	Bank Select
FFD8	C-REG-SEL	Colour register
FFE0	KYBD REG	Keyboard register

B.2 PIA - $FFC0 - PARALLEL I/O

B.2.1 The PIA is used for the centronics port and for bank switching the additional memory cards. Details of its operation may be found in the HD46821P technical data sheet.

B.3 ACIA - $FFC4 - SERIAL I/O

B.3.1 The ACIA is used for RS-232 communication via the serial port (COM0, etc.) and for CAS0. This is why COMx and CAS0 may not be used concurrently. Details on programming the ACIA may also be found in the HD46850P technical data sheet.

B.4 CRTC - $FFC6 - CRT CONTROLLER

B.4.1 Full details on the operation of the CRTC may be found in the HD46505SP technical data sheet. The following information relates to the initial values placed in the CRTC registers on initialisation. The following values relate to 40 and 80 column mode with interlace and non-interlace values for registers 8 to 13:-

REG.	NAME	40 COL.	80 COL.
R0	Horizontal Total	3F	7F
R1	Horizontal Displayed	28	50
R2	H. Sync Position	31	5F
R3	H. Sync Width	85	8A
R4	Vertical Total	1F	1F
R5	V. Total Adjust	06	06
R6	Vertical Displayed	19	19
R7	V. Sync Position	1C	1C
		INT MODE	NON-INT
R8	Interlace Mode	83	80
R9	Max. Scan Line Addr.	0E	07
R10	Cursor Start	2E	27
R11	Cursor End	0F	07
R12	Start Address (H)	04*	04*
R13	Start Address (L)	00*	00*
R14	Cursor (H)		
R15	Cursor (L)		
R16	Light Pen (H)		
R17	Light Pen (L)		

* - this value is the start of the screen display and varies depending on the screen page being displayed

B.4.2 Some experimentation with the above values may assist with removing the instability that has been apparent with some monitors. Try setting R4 to $24 and R5 to between $12 and $14.

B.5 KBNMI - $FFC8 - KEYBOARD NMI FLAG

B.5.1 Bit 7 of this register is set when the BREAK key is pressed. It remains set while the key is held down and may be cleared by reading the register.

B.6 DIPSW - $FFC9 - SWITCH SETTINGS

B.6.1 Reading this register gives the physical switch settings. A copy of this value is stored in location $0248.

B.7 TIMER - $FFCA - TIMER FLAG

B.7.1 Bit 7 of this register is set when a timer IRQ interrupt is generated. The register is cleared when read.

B.8 LPFLG - $FFCB - L/PEN FLAG

B.8.1 Bit 7 of this register is set when a Light Pen IRQ interrupt is generated. The register is cleared when read.

B.9 MODE SEL - $FFD0 - SCREEN MODE, ETC.

B.9.1 The following bit settings define the screen mode, RS-232/Cassete enable and the screen background colour:-

```
  7 - 0=40 ch.   1=80 ch.
  6 - 0=High res.   1=Low res.
  5 - 0=Cassette   1=RS-232
2-0 - Background colour
```


B.10 TRACE - $FFD1 - TRACE COUNTER

B.10.1 This register appears to provide a single step facility. A write to the register will cause a NMI interrupt after 15 CPU cycles. This would be long enough to execute an RTI instruction and have the interrupt occur during the next instruction. The interrupted instruction would be completed before the interrupt was serviced. This register is used for the single step facility in the Hitachi debugger (DEBUG5A and DEBUG70).

B.11 MOTOR - $FFD2 - CASSETTE MOTOR ON/OFF

B.11.1 Setting bit 7 of this register turns the cassette motor relay on. A copy of the current setting is stored in location $01F7.

B.12 MUSIC SEL - $FFD3 - MUSIC SELECT

B.12.1 Varying the top bit setting of this register generates different sounds.

B.13 TIME MASK - $FFD4 - TIMER ENABLE

B.13.1 Setting bit 7 of this register turns off the timer IRQ interrupt signal.

B.14 L/PEN MASK - $FFD5 - LIGHT PEN ENABLE

B.14.1 Setting bit 7 of this register enables the Light Pen IRQ interrupt signal.

B.15 INT'LCE SEL - $FFD6 - INTERLACE SELECT

B.15.1 Setting bit 3 of this register enables interlace mode and zeroising it disables interlace mode.

B.16 C-REG-SEL - $FFD8 - COLOUR REGISTER

B.16.1 The following bit setings define various screen attributes. Bits 0 to 4 are the same as stored in the special 5 bit wide colour RAM:-

```
  7 - 0=Enable write to special colour RAM
      1=Prohibit write to special colour RAM
  4 - 0=Character  1=Graphic
  3 - 0=Normal  1=Inverse Video
2-0 - Colour (same value as BASIC COLOR statement)
```

B.17 KYBD REG - $FFE0 - KEYBOARD REGISTER

B.17.1 This register has different functions when written to or read from. On reading it returns the current keyboard scan value. This defines the key that was pressed and generated a keyboard IRQ interrupt request. The values for each key are defined in Appendix D and are decoded by the keyboard interrupt routine into ASCII character values. The last $FFE0 value may be found in location [01CA]+1 and is useful for determining the last key pressed ignoring all the characters in the keyboard advance buffer.

B.17.2 The values written to this register are as follows. A copy of the current value written is stored in location [01CA]+$3:-

```
7 - 0=KYBD NMI disable  1=KYBD NMI enable
6 - 0=KYBD IRQ disable  1=KYBD IRQ enable
3 - 0=KYBD counter enable  1=KYBD counter disable
2 - 0=SHIFT light on  1=SHIFT light off
1 - 0=HIRA light off  1=HIRA light on
0 - 0=KATA light off  1=KATA light on
```

B.17.3 Setting bit 3 to '1' is one method of turning the keyboard advance buffer off (refer Section 4.4 for further details).

C.1 BASIC JUMP TABLE

INSTRUCTION	TOKEN	ROM JUMP	COMMENTS
ABS	FF 82	C473	
AND	D3	--	
ASC	FF 90	AF01	
ATN	FF 8B	CF66	
AUTO	9D	B42C	
BEEP	B4	F7A2	
CDBL	FF 94	CD5E	
CHR$	FF 91	AEED	
CINT	FF 92	CCF7	
CLEAR	BC	DE7C	
CLOSE	AF	E61A	
CLS	A4	F160	
COLOR	B5	DFA2	
COM ON/OFF/STOP	B1	EBD8	ON-EBF4 OFF-EBE5 STOP-EC03
CONSOLE	A5	DFCF	
CONT	BA	A5AD	
COS	FF 88	CED0	
CSNG	FF 93	CD71	
CSRLIN	FF A6	E07C	
DATA	83	A647	
DATE (...=DATE)	FF A9	D970	
DATE$ (DATE$=...)	FF A9	D9C4	Assignment of value to DATE$
DATE$ (...=DATE$)	FF A9	D95F	
DEF FN	B7/C4	BB84	
DEF USR	B7/C7	BD18	
DEFDBL	96	B3BA	
DEFINT	94	B3B4	
DEFSNG	95	B3B7	
DEFSTR	93	B3B1	
DELETE	9E	B3FA	
DIM	84	AA55	

EDIT	9A	F01F
ELSE	8F	A64A
END	80	A57B
EOF	FF 9A	E89B
EQV	D6	--
ERL	C8	--
ERR	C9	--
ERROR	9B	BB02
EXP	FF 87	CBD4
EXEC	AD	DE70
FILES	B0	E39E
FIX	FF 95	C4FF
FOR	81	B610
FRE	FF 83	ACC8
GOSUB	87/C3	A5D1
GOTO	87/C2	A5D1
HEX$	FF 97	B25F
IF	89	A67B
IMP	D7	--
INKEY$	FF A4	F868
INPUT	FF A5	D06C
INPUT$	FF A5	E8DD
INPUT WAIT	FF A5/98	D047
INSTR	FF 9F	B11E
INT	FF 81	C50F
KEY	B2	FA0D
KEY LIST	B2/BB	F9D1
KEY ON/OFF/STOP	BB	FA07
LEFT$	FF 9C	AF0C
LEN	FF 8D	AEE4
LET	86	A71D
LINE	B6	D67F
LINE INPUT	B6/FF A5	CFBA
LIST	BB	E4BE
LOAD	AB	E6D4

LOAD?	AB	E2CF	
LOADM	AB	E6D4	
LOCATE	A3	E04F	
LOF	FF 99	E89E	
LOG	FF 86	C15D	
MERGE	AC	E6CC	
MID$ (MID$=...)	FF 9E	B1D9	
MID$ (...=MID$)	FF 9E	AF36	
MOD	D8	--	
MON	A2	DCE6	
MOTOR	A8	EE3C	
NEW	C0	A4FE	
NEW ON	C0/97	A4F0	FAB3 with NEW ON value in A
NEXT	82	B6E1	
NOT	CC	--	
OCT$	FF 98	B260	
ON COM	97/B1	EB9B	
ON ERROR	97/9B	BADA	
ON GOSUB	97/87/C3	A6BB	
ON GOTO	97/87/C2	A6BB	
ON KEY	97/B2	FA87	
ON PEN	97/FF 9B	E1B4	
OPEN	AE	E64E	
OR	D4	--	
PAINT	B3	DB74	
PEEK	FF 8C	AFDF	
PEN	FF 9B	E16D	Defn. of PEN screen area
PEN (...=PEN)	FF 9B	E09F	PEN function
PEN ON/OFF/STOP	FF 9B	E160	
POINT	FF A7	D577	
POKE	B8	AFE9	
POS	FF 84	E082	
PRESET	A7	D599	
PRINT	B9	B043	
PRINT #	B9	B04C	

PRINT USING	B9/C6	B4BC	
PSET	A6	D595	
RANDOMIZE	BD	CC63	
READ	85	D091	
REM	8C	A64A	
RENUM	99	BD6C	
RESTORE	8A	A567	
RESUME	9C	BB0D	
RETURN	8B	A614	
RIGHT$	FF 9D	AF2C	
RND	FF A3	CC7C	
RUN	88	A5BD	
SAVE	AA	E4FA	
SAVEM	AA	E5AD	
SCREEN (SCREEN=.)	FF A0	DEB2	
SCREEN (.=SCREEN)	FF A0	D840	SCREEN function
SGN	FF 80	C45C	
SIN	FF 89	CED6	
SKIPF	A9	E2D1	
SPACE$	FF 96	B1C3	
SPC(	C5	--	
SQR	FF 85	CB67	
STEP	CD	--	
STOP	8E	A584	
STR$	FF 8E	ACF2	
STRING$	FF A2	B1A4	
SWAP	92	BB4A	
TAB(	C1	--	
TAN	FF 8A	CF22	
TERM	9F	EC49	
THEN	CB	--	
TIME (...=TIME)	FF A8	D8F7	TIME function
TIME$ (TIME$=...)	FF A8	D9B0	Assignment of value to TIME$
TIME$ (...=TIME$)	FF A8	D939	
TROFF	91	B45F	
TRON	90	B45E	

USR	C7	--	
VAL	FF 8F	AF8F	
VARPTR	FF A1	BDOD	
WAIT	98	AFF3	Refer Section 6.64
WIDTH	AO	E012	
WEND	BF	017F	)Gives 'Syntax
WHILE	BE	017C	)Error'

OPERATORS

' (Remark)	8D	A64A
+ (Addition)	CE	--
- (Subtraction)	CF	--
* (Multiplication)	DO	--
/ (Division)	D1	--
^ (Exponentiation)	D2	--
¥ (Integer Div.)	D9	--
> (Greater than)	DA	--
= (Equals)	DB	--
< (Less than)	DC	--

C.2 JUMPS FOR SPECIAL CHARACTERS

CHARACTER	HEX VALUE	JUMP
One Word Left	02	F48B
Clear Line	05	F415
One Word Right	06	F47F
Bell	07	F7A7
Delete	08	F3AF
Horizontal Tab	09	F283
Line Feed	0A	F124
Home	0B	F17B
Clear Screen	0C	F160
Carriage Return	0D	F180
Insert	12	F2FA
Set Tab	.14	F260
Display Cursor	16	F2AA
Clear Cursor	17	F2AE
Clear Tab	19	F267
Clear to Bottom	1A	F403
Right Arrow	1C	F18E
Left Arrow	1D	F19D
Up Arrow	1E	F189
Down Arrow	1F	F194

C.3 BASIC TOKENS IN NUMERICAL ORDER

80	END	9F	TERM	BE	WHILE
81	FOR	A0	WIDTH	BF	WEND
82	NEXT	A1	UNLIST	C0	NEW
83	DATA	A2	MON	C1	TAB(
84	DIM	A3	LOCATE	C2	TO
85	READ	A4	CLS	C3	SUB
86	LET	A5	CONSOLE	C4	FN
87	GO	A6	PSET	C5	SPC
88	RUN	A7	PRESET	C6	USING
89	IF	A8	MOTOR	C7	USR
8A	RESTORE	A9	SKIPF	C8	ERL
8B	RETURN	AA	SAVE	C9	ERR
8C	REM	AB	LOAD	CA	OFF
8D	'	AC	MERGE	CB	THEN
8E	STOP	AD	EXEC	CC	NOT
8F	ELSE	AE	OPEN	CD	STEP
90	TRON	AF	CLOSE	CE	+
91	TROFF	B0	FILES	CF	-
92	SWAP	B1	COM	D0	*
93	DEFSTR	B2	KEY	D1	/
94	DEFINT	B3	PAINT	D2	^
95	DEFSNG	B4	BEEP	D3	AND
96	DEFDBL	B5	COLOR	D4	OR
97	ON	B6	LINE	D5	XOR
98	WAIT	B7	DEF	D6	EQV
99	RENUM	B8	POKE	D7	IMP
9A	EDIT	B9	PRINT	D8	MOD
9B	ERROR	BA	CONT	D9	¥
9C	RESUME	BB	LIST	DA	>
9D	AUTO	BC	CLEAR	DB	=
9E	DELETE	BD	RANDOMIZE	DC	<

C.4 TOKENS FOR BASIC FUNCTIONS

FF 80	SGN	FF 95	FIX
FF 81	INT	FF 96	SPACE$
FF 82	ABS	FF 97	HEX$
FF 83	FRE	FF 98	OCT$
FF 84	POS	FF 99	LOF
FF 85	SQR	FF 9A	EOF
FF 86	LOG	FF 9B	PEN
FF 87	EXP	FF 9C	LEFT$
FF 88	COS	FF 9D	RIGHT$
FF 89	SIN	FF 9E	MID$
FF 8A	TAN	FF 9F	INSTR
FF 8B	ATN	FF A0	SCREEN
FF 8C	PEEK	FF A1	VARPTR
FF 8D	LEN	FF A2	STRING$
FF 8E	STR$	FF A3	RND
FF 8F	VAL	FF A4	INKEY$
FF 90	ASC	FF A5	INPUT
FF 91	CHR$	FF A6	CSRLIN
FF 92	CINT	FF A7	POINT
FF 93	CSNG	FF A8	TIME
FF 94	CDBL	FF A9	DATE

D.1 INTRODUCTION

D.1.1 The following tables document the keyboard. TABLE 1 contains a list of the keys and the hex values generated by each key alone and in combination with the various shift keys. TABLE 1 also documents the values returned for each key in the hardware keyboard register at $FFE0. TABLE 2 contains a list of all possible byte values, and the keys or combination of keys which can be used to input each value. The ASCII definitions and the BASIC control functions are included.

D.1.2 Later models of the Hitach MB6890 sold in Australia do not have the YEN key or the HIRA/KATA key.

D.2 TABLE 1 - KEYBOARD TRANSLATIONS

KEY NO — The keys are numbered from left to right, top to bottom, starting with the main block of keys, then the BREAK and PF keys and finally the numeric keypad.

KEY LEGEND — The legend on each key is in the second column of the table.

COUNT — The third column is a count generated by the keyboard hardware whenever the key is pressed. The count is read from $FFE0 and stored at [$01CA]+1 by the key input interrupt routine.

Format — unshifted hex value/shifted hex value
'-' indicates no hex value returned

The remaining columns contain the hex values returned to a BASIC program, or returned by the $E804 routine, for each key. Each column is in the format "unshifted value/shifted value". The unshifted value is the value returned with CAPS LOCK not selected (red indicator off) and neither SHIFT key depressed. This is the normal mode for BASIC. The shifted value is returned when CAPS LOCK is not selected (red indicator off) and one of the SHIFT keys is depressed. If both unshifted and shifted value are the same only one value is displayed in a column. A dash (-) indicates that no value is returned.

PRIME FN — The fourth column contains the values returned by the primary function of each key.

KATA — The fifth column contains the values returned when KATAKANA is selected (red indicator on).

HIRA — The sixth column contains the values returned when HIRAGANA is selected (green indicator on).

GRAPH — The seventh column contains the value returned when the GRAPH key is depressed.

CTRL — The last column contains the value returned when CTRL is depressed.

KEY NO	KEY LEGEND	COUNT ($FFE0)	PRIME FN	KATA	HIRA	GRAPH	CTRL
1	ESC	8C	1B	1B	1B	1B	-
2	1/!	9A	31/21	C7	E7	A0	-
3	2/"	9B	32/22	CC	EC	F2/F3F2	-
4	3/#	97	33/23	B1/A7	91/87	F3/F3F2	-
5	4/$	91	34/24	B3/A9	93/89	F4	-
6	5/%	99	35/25	B4/AA	94/8A	F5	-
7	6/&	92	36/26	B5/AB	95/8B	F6	-
8	7/'	90	37/27	D4/AC	F4/8C	F7	-
9	8/(	93	38/28	D5/AD	F5/8D	8C	-
10	9/)	9C	39/29	D6/AE	F6/8E	F9	-
11	0	94	30	DC/A6	FC/86	F9	-
12	-/=	96	2D/3D	CE	EE	E0	-
13	^	95	5E	CD	ED	FB	1E
14	¥	9F	5C	B0/90	B0	F1	1C
15	DEL	9E	08/12	08/12	08/12	-	-
16	HIRA/KATA	8A	-	-	-	-	-
17	Q	A8	51/71	C0	E0	E8	11
18	W	AA	57/77	C3	E3	EA	17
19	E	AB	45/65	B2/A8	92/88	EC/F0EC	05
20	R	A1	52/72	BD	9D	9C	12
21	T	A9	54/74	B6	96	9D	14
22	Y	A2	59/79	DD	FD	E4	19
23	U	A0	55/75	C5	E5	E5	15
24	I	A3	49/69	C6	E6	8D	09
25	O	AC	4F/6F	D7	F7	8A	0F
26	P	A4	50/70	BE	9E	FA	10
27	@	A6	40	DE	DE	E1	00
28	[	A5	5B/-	DF/A2	DF/A2	FC	1B
29(1)	RETURN	AF	0D	0D	0D	-	-

KEY NO	KEY LEGEND	COUNT ($FFE0)	PRIME FN	KATA	HIRA	GRAPH	CTRL
30	CAPS LOCK	89	-	-	-	-	-
31	CTRL	86	-	-	-	-	-
32	A	B8	41/61	C1	E1	E9	01
33	S	BA	53/73	C4	E4	EB	13(2)
34	D	BB	44/64	BC	9C	ED/93ED	04(3)
35	F	B1	46/66	CA	EA	9E	06
36	G	B9	47/67	B7	97	9F	07
37	H	B2	48/68	B8	98	E6	08
38	J	B0	4A/6A	CF	EF	E7	0A
39	K	B3	4B/6B	C9	E9	8E	0B
40	L	BC	4C/6C	D8	F8	89	0C
41	;/+	B4	3B/2B	DA	FA	94	-
42	:/*	B6	3A/2A	B9	99	E2	-
43	]	B5	5D/-(5)	D1/A3	F1/A3	FE	1D
44	GRAPH	8B	-	-	-	-	-
45(1)	SHIFT	87	-	-	-	-	-
46	Z	C8	5A/7A	C2/AF	E2/8F	80	1A
47	X	CA	58/78	BB	9B	81	18
48	C	CB	43/63	BF	9F	82/9482	03(4)
49	V	C1	56/76	CB	EB	83	16
50	B	C9	42/62	BA	9A	84	02
51	N	C2	4E/6E	D0	F0	85	0E
52	M	C0	4D/6D	D3	F3	86	0D
53	,/<	C3	2C/3C	C8/A4	E8/A4	87	-
54	./>	CC	2E/3E	D9/A1	F9/A1	88	-
55	//?	C4	2F/3F	D2/A5	F2/A5	97	-
56	_	C6	5F	DB	FB	E3	1F
57(1)	SHIFT	87	-	-	-	-	-
58	SPACE	80	20	20	20	20	-

KEY NO	KEY LEGEND	COUNT ($FFEO)	PRIME FN	KATA	HIRA	GRAPH	CTRL
60	BREAK	-	-	-	-	-	-
61	PF1	D0	-	-	-	-	-
62	PF2	D1	-	-	-	-	-
63	PF3	D2	-	-	-	-	-
64	PF4	D3	-	-	-	-	-
65	PF5	D4	-	-	-	-	-

NUMERIC PAD KEYS

66	CLS/HOME	AE	0C/0B	0C/0B	0C/0B	-	-
67	↑	81	1E	1E	1E	1E	-
68	?	82	3F	3F	3F	3F	-
69	DLE/INS	98	08/12	08/12	08/12	-	-
70	←	83	1D/02	1D/02	1D/02	1D/02	-
71	↓	84	1F	1F	1F	1F	-
72	→	85	1C/06	1C/06	1C/06	1C/06	-
73	/	C5	2F	2F	2F	EE	-
74	7	9D	37	37	37	98	-
75	8	8D	38	38	38	91	-
76	9	8E	39	39	39	99	-
77	*	8F	2A	2A	2A	EF	-
78	4	B7	34	34	34	93/93ED	-
79	5	BD	35	35	35	8F	-
80	6	BE	36	36	36	92	-
81	-	BF	2D	2D	2D	95	-
82	1	C7	31	31	31	9A/9A82	-
83	2	CD	32	32	32	90	-
84	3	CE	33	33	33	9B	-
85	+	CF	2B	2B	2B	96	-
86	0	A7	30	30	30	F0/F0EC	-
87	.	AD	2E	2E	2E	F8	-
88(1)	RETURN	AF	0D	0D	0D	-	-

NOTE - refer Section D.4 for notes on TABLE 1

D.3 TABLE 2 - KEYBOARD HEX AND ASCII VALUES

D.3.1 The following table contains all possible byte values ($00 to $FF) and the keys, or combination of keys, which can be used to input each value from the keyboard. The ASCII definitions and the BASIC control functions are included.

HEX — Hex byte values are listed in order from $00 to $FF.

ASCII — ASCII definitions for values $00 to $FF are listed.

KEYS — The keys, or key combinations, for which the hex value is returned to a BASIC program, or by the $E804 routine are listed for each all values.

FUNCTION — The BASIC functions for the ASCII control characters, $00 to $1F, are listed.

CONTROL CHARACTERS

HEX	ASCII	KEYS		FUNCTION
00	nul	ctrl @		
01	soh	ctrl A		
02	stx	ctrl B	shft ←	Cursor to word left
03(4)	etx	ctrl C		(4)Break BASIC program
04(3)	eot	ctrl D		Break BASIC program
05	enq	ctrl E		Erase to end of line
06	ack	ctrl F	shft →	Cursor to word right
07	bel	ctrl G		Sound alarm
08	bs	ctrl H	DEL	Delete character left
09	ht	ctrl I		Tab right
0A	lf	ctrl J		
0B	vt	ctrl K	HOME(shft CLS)	Cursor home
0C	ff	ctrl L	CLS	Clear screen
0D	cr	ctrl M	RETURN	New line
0E	so	ctrl N		
0F	si	ctrl O		
10	dle	ctrl P		
11	dc1	ctrl Q		
12	dc2	ctrl R	INS(shft DEL)	Insert space
13(2)	dc3	ctrl S		(2)Suspend BASIC program
14	dc4	ctrl T		Set tab
15	nak	ctrl U		
16	syn	ctrl V		
17	etb	ctrl W		
18	can	ctrl X		
19	em	ctrl Y		Clear tab
1A	sub	ctrl Z		Erase to end of screen
1B	esc	ctrl [	ESC	
1C	fs	ctrl ¥	→	Cursor right one space
1D	gs	ctrl]	←	Cursor left one space
1E	rs	ctrl ^	↑	Cursor up one line
1F	us	ctrl _	↓	Cursor down one line

DISPLAY CHARACTERS

HEX	ASCII	KEYS	
20	sp	space bar	
21	!	!(shft 1)	
22	"	"(shft 2)	
23	#	#(shft 3)	
24	$	$(shft 4)	
25	%	%(shft 5)	
26	&	&(shft 6)	
27	'	'(shft 7)	
28	(	((shft 8)	
29	)	)(shft 9)	
2A	*	*(shft :)	pad *
2B	+	+(shft ;)	pad +
2C	,	,	
2D	-	-	pad -
2E	.	.	pad .
2F	/	/	pad /
30	0	0	pad 0
31	1	1	pad 1
32	2	2	pad 2
33	3	3	pad 3
34	4	4	pad 4
35	5	5	pad 5
36	6	6	pad 6
37	7	7	pad 7
38	8	8	pad 8
39	9	9	pad 9
3A	:	:	
3B	;	;	
3C	<	<(shft ,)	
3D	=	=(shft -)	
3E	>	>(shft .)	
3F	?	?(shft /)	pad ?

ALPHABET CHARACTERS

HEX	ASCII	KEY	HEX	ASCII	KEY
40	@	@	60(7,8)	`	
41	A	A	61	a	shft A
42	B	B	62	b	shft B
43	C	C	63	c	shft C
44	D	D	64	d	shft D
45	E	E	65	e	shft E
46	F	F	66	f	shft F
47	G	G	67	g	shft G
48	H	H	68	h	shft H
49	I	I	69	i	shft I
4A	J	J	6A	j	shft J
4B	K	K	6B	k	shft K
4C	L	L	6C	l	shft L
4D	M	M	6D	m	shft M
4E	N	N	6E	n	shft N
4F	O	O	6F	o	shft O
50	P	P	70	p	shft P
51	Q	Q	71	q	shft Q
52	R	R	72	r	shft R
53	S	S	73	s	shft S
54	T	T	74	t	shft T
55	U	U	75	u	shft U
56	V	V	76	v	shft V
57	W	W	77	w	shft W
58	X	X	78	x	shft X
59	Y	Y	79	y	shft Y
5A	Z	Z	7A	z	shft Z
5B	[	[	7B(7)	{	
5C(6)	\	¥	7C(7)	¦	
5D	]	]	7D(7)	}	
5E	^	^	7E(7)	tilde	
5F	_	_	7F(6,7)	del	

HEX	KEYS: GRAPH	KEYS: HIRA+shft
80	Z	
81	X	
82	C	
83	V	
84	B	
85	N	
86	M	0
87	,	3
88	.	E
89	L	4
8A	0	5
8B	9	6
8C	8	7
8D	I	8
8E	K	9
8F	pad 5	Z

HEX	KEYS	HIRA	KATA+shft
90	pad 2		¥
91	pad 8	3	
92	pad 6	E	
93	pad 4	4	
94	;	5	
95	pad -	6	
96	pad +	T	
97	/	G	
98	pad 7	H	
99	pad 9	:	
9A	pad 1	B	
9B	pad 3	X	
9C	R	D	
9D	T	R	
9E	F	P	
9F	G	C	

HEX	KEYS: KATA+shft	HIRA+shft	GRAPH
A0			1
A1	.	.	
A2	[	[	
A3	]	]	
A4	,	,	
A5	/	/	
A6	0		
A7	3		
A8	E		
A9	4		
AA	5		
AB	6		
AC	7		
AD	8		
AE	9		
AF	Z		

HEX	KATA	HIRA
B0	¥	¥
B1	3	
B2	E	
B3	4	
B4	5	
B5	6	
B6	T	
B7	G	
B8	H	
B9	:	
BA	B	
BB	X	
BC	D	
BD	R	
BE	P	
BF	C	

HEX	KEYS	
	KATA	HIRA
C0	Q	
C1	A	
C2	Z	
C3	W	
C4	S	
C5	U	
C6	I	
C7	1	
C8	,	
C9	K	
CA	F	
CB	V	
CC	2	
CD	^	
CE	-	
CF	J	
	KATA	HIRA
D0	N	
D1	]	
D2	/	
D3	M	
D4	7	
D5	8	
D6	9	
D7	0	
D8	L	
D9	.	
DA	;	
DB		
DC	$\overline{0}$	
DD	Y	
DE	@	@
DF	[	[

HEX	KEYS	
	HIRA	GRAPH
E0	Q	-
E1	A	@
E2	Z	:
E3	W	
E4	S	$\overline{Y}$
E5	U	U
E6	I	H
E7	1	J
E8	,	Q
E9	K	A
EA	F	W
EB	V	S
EC	2	E
ED	^	D
EE	-	pad /
EF	J	pad *
	HIRA	GRAPH
F0	N	pad 0
F1	]	¥
F2	'	2
F3	M	3
F4	7	4
F5	8	5
F6	9	6
F7	0	7
F8	L	pad .
F9	.	0
FA	;	P
FB		^
FC	$\overline{0}$	[
FD	Y	
FE		]
FF(9)		

D.4 NOTES ON APPENDIX D

(1) There is no way for a program to determine which of the two RETURN keys, or which of the two SHIFT keys have been pressed.

(2) The values for CTRL-S ($13) is only returned to a BASIC program or by the $E804 routine when $03D6 is non zero. The BASIC suspend function is then disabled.

(3) The value for CTRL-D ($04) is never returned to a BASIC program. Once CTRL-D is entered $04 is returned for every subsequent call to the $E804 routine and no other input is accepted.

(4) When $03D6 is zero CTRL-C is handled in the same way as CTRL-D. When $03D6 is non zero CTRL-C is treated as a normal character and the BASIC BREAK function is disabled.

(5) Shift] controls the screen background colour.

(6) The ASCII del character ($7F) is displayed as a backslash by the Hitachi MB6890. The del character is not normally displayed. The backslash character ($5C) is displayed as a Yen sign (¥) on the Hitachi MB6890.

(7) The following characters can only be input from the keyboard by redefining the PF keys.

HEX	ASCII
60	`
7B	{
7C	|
7D	}
7F	tilde
7F	del

(8) The grave character ($60) is displayed as a space by the Hitachi MB6890.

(9) Once the BREAK key in pressed, $FF is returned for every subsequent call to the $E804 routine and no other input is accepted.

E.1 INTRODUCTION

E.1.1 The following notes relate specifically to SSSD Disk Basic. Reference to Disk Basic has been avoided, where possible, in the other sections of the notes to avoid any confusion between different versions of Disk Basic. In addition, all the experimentation with the ROM has been undertaken using SSSD Disk Basic. It is possible that other versions of Disk Basic may behave differently under certain conditions.

E.1.2 These notes are limited in scope and are intended to complement other areas of the notes. They are not intended as a detailed explanation of how SSSD Disk Basic works.

E.1.3 All references to addresses relate to the 32K version of SSSD Disk Basic. To convert the addresses for 40K Disk Basic $2000 should be added to every address, i.e., the first digit, in Hex, should be changed from 7 to 9.

E.2 SSSD DISK BASIC JUMP TABLE

INSTRUCTION	TOKEN	ROM JUMP	COMMENTS
COMMANDS			
DSKINI	DD	7411	
DSKO$	DE	7620	
KILL	DF	7674	
NAME	E0	7C3E	
FIELD	E1	7C95	
LSET	E2	7CD3	
RSET	E3	7CD2	
PUT	E4	7D2C	
GET	E5	7D2F	

FUNCTIONS

DSKF	FF AA	7B75
CVI	FF AB	7B9D
CVS	FF AC	7BA0
CVD	FF AD	7BA3
MKI$	FF AE	7BB7
MKS$	FF AF	7BBA
MKD$	FF B0	7BBD
LOC	FF B1	7BD4
DSKI$	FF B2	7660

DOS JUMPS

FILES	7AF5
OPEN	77A4
CLOSE	78BD
$E820 Character Output	7918
$E804 Character Input	79AE
$E84B ??	7A50
EOF/LOF	7C05
INPUT # ??	7A5F
Sector I/O	7508

E.3 SSSD DISK BASIC WORK AREA

72BC	Disk file buffer pointers
72DA	Highest drive no. - set by CONFIG
72DB	Maximum no. of disk files that may be open at any one time - set by CONFIG
72DC-72DD	Address for sector I/O data (DSKI and DSKO)
72DE	Drive no. for I/O
72DF	I/O mode - 02=Read 03=Write
72E0	Sector no. for I/O
72E2	Track no. for I/O

APPENDIX E - NOTES ON SSSD DISK BASIC

E.4 DISK I/O ROUTINES

E.4.1 The basic Disk I/O routines relate to sector I/O, i.e., read or write a sector. Character I/O may be executed by using the standard $E804 and $E820 character I/O routines with the file opened for disk.

- JSR $7508 - Sector I/O routine - set locations $72DC-72E2 as appropriate
- JSR $762D - Set data pointer in the X register and the I/O mode in the B register
- JSR $766D - Read a sector - data pointer in X register
- JSR $762B - Write a sector - data pointer in X register

E.5 LIMITED NOTES ON HOW DISK BASIC WORKS

E.5.1 Disk Basic is essentially similar to cassette Basic as far as its basic processing cycle is concerned. It uses the ROM routines for processing not related to disk.

E.5.2 The ROM automatically attempts to boot from disk as part of its power-on initialisation. The disk boot routine starts at location $FE7E. The disk bootstrap loader is located in the first two sectors of track zero. The ROM tests for the disk controller card and attempts to read in these two sectors. If the disk is not ready then the ROM continues with the cassette initialisation. The bootstrap loader is loaded into locations $4400-44FF but this area is set back to zero before passing control to BASIC.

E.5.3 The Disk Basic code is located in tracks one and two and is loaded into location \$7000 onwards (or \$9000 for 40K Disk Basic). Locations \$7000-72B7 contain initialisation code and is overwritten by the BASIC string area and the stack when control is passed to BASIC. The Disk Basic code starts at location \$72B8 (or \$92B8 for 40K Disk Basic). The code ends at location \$7F22 and the remining bytes to \$7FFF are not used. The DOSMOD program detailed in Appendix F uses this area.

F.1 INTRODUCTION

F.1.1 This section details two programs that are available to purchasers of the notes. The details for obtaining these programs is included Section 1.7.

F.2 MEMORY DISPLAY ROUTINE

F.2.1 This routine allows memory bytes to be displayed on the screen concurrently with other processing. BASIC and machine programs may be run and normal editing functions may be carried out. This routine will allow changes to specified areas of memory to be observed during normal processing and is exceptionally useful for determining the use of bytes in the BASIC and system work area (below the start of BASIC).

F.2.1 The routine is implemented as part of the timer interrupt processing. It displays the specified range of memory bytes sixty times per second. It passes the interrupt to the normal routine at the completion of processing. Its only effect is to slow down the processing including BASIC's TIME function. A limited number of memory may be displayed when using CASO or COMO. This is not considered restrictive as it is only intended as an experimentation routine and should not be used during "production" processing.

F.2.2 Further details on the running of this program will be supplied with the code. Both the assembler source code and the object code will be supplied (the assembler source has been written using the Hitachi assembler ASM9).

F.3 SSSD DOSMOD PROGRAM

F.3.1 This program modifies SSSD Disk Basic, both 32K and 40K versions, to fix several problems and improve the FILES display. The program modifies the code on the disk and does not modify the code in memory - the system must be re-booted to implement the modifications. At least one unmodified copy of the DOS should be kept in case the modifications cause problems.

F.3.2 The code has been specifically tested to ensure it is working correctly. In addition it has been in use by several Peach owners for some months without any problems. No guarantee is given that the code is totally error free, howver, it is unlikely to cause problems.

F.3.3 The modifications are as follows:-

1 - Amends 40K Disk Basic to automatically execute the program with a blank file name on initialisation. This is similar to the feature available with 32K Disk Basic. Refer Section 8.9 for further details

2 - Amends 32K Disk Basic to execute the same code as 40K Disk Basic. This fixes the problem with 32K Disk Basic where, if no file with a blank file name is present, the system boots up in COLOR 0,0 (black on a black background). This modified code will generate a "File Not Found" message if no file with a blank file name is present

3 - Amends both 32K and 40K Disk Basic to print 'FILES' across the screen instead of down the screen. This fixes the problem with the screen automatically scrolling after 22 files. In 80 column mode four columns of files are displayed and in 40 column mode two columns of files are displayed. The format of the 'FILES' data is not changed

4 - Amends both 32K and 40K Disk Basic to fix the problem with the "NOT EOF(x)" test for disk files. Refer Section 8.8 for further details

5 - Amends both 32K and 40K Disk Basic to fix a potential problem with the 'FILES' command. Refer Section 8.12 for details. This modification is optional as it will not allow the 'FILES' command to be excuted if a file is open on the same drive (the disk may be inserted in another drive and 'FILES' executed). The program prints a prompt to allow or ignore the modification.

F.3.4 The program may be run against an already modified disk as the code checks for both the unmodified and modified bytes. For option 5 (the 'FILES' command restriction) the program may be run to either modify the DOS or to return it to the original code. If the prompt message specifies that the modification is to be ignored and the code is already modified then the code is re-modified to the original values. Therefore, this modification may be experimented with to see if the restriction causes problems.

F.3.5 The program checks a few bytes to ensure the disk does actually contain the Disk Basic code. It also checks the FAT on the directory track to ensure the first three tracks are reserved. If the disk does not appear to contain Disk Basic code an error message will be printed together with the status of modifications already performed. It is exceptionally unlikely that the program would modify a disk that did not contain Disk Basic code.